

FLASH® LITE™ 2.x ACTIONSCRIPT™ リファレンスガイド

© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Lite™ 2.x ActionScript™ リファレンスガイド

本マニュアルがエンドユーザー使用許諾契約を含むソフトウェアと共に提供される場合、本マニュアルおよびその中に記載されているソフトウェアは、エンドユーザー使用許諾契約にもとづいて提供されるものであり、当該エンドユーザー使用許諾契約の契約条件に従ってのみ使用または複製することが可能となるものです。当該エンドユーザー使用許諾契約により許可されている場合を除き、本マニュアルのいかなる部分といえども、Adobe Systems Incorporated（アドビ システムズ社）の書面による事前の許可なしに、電子的、機械的、録音、その他いかなる形式・手段であれ、複製、検索システムへの保存、または伝送を行うことはできません。本マニュアルの内容は、エンドユーザー使用許諾契約を含むソフトウェアと共に提供されていない場合であっても、著作権法により保護されていることにご留意ください。

本マニュアルに記載される内容は、あくまでも参考用としてのみ使用されること、また、なんら予告なしに変更されることを条件として、提供されるものであり、従って、当該情報が、アドビ システムズ社による確約として解釈されることはなりません。アドビ システムズ社は、本マニュアルにおけるいかなる誤りまたは不正確な記述に対しても、いかなる義務や責任を負うものではありません。

新しいアートワークを創作するためにテンプレートとして取り込もうとする既存のアートワークまたは画像は、著作権法により保護されている可能性のあるものであることをご留意ください。保護されているアートワークまたは画像を新しいアートワークに許可なく取り込んだ場合、著作権者の権利を侵害することがあります。従って、著作権者から必要なすべての許可を必ず取得してください。

例として使用されている会社名は、実在の会社・組織を示すものではありません。

Adobe、Adobe ロゴ、Flash Lite および Flash は、アドビ システムズ社の米国ならびに他の国における商標または登録商標です。

サードパーティ情報

本マニュアルには、アドビ システムズ社が管理していない、サードパーティの Web サイトへのリンクが掲載されていますが、アドビ システムズ社はいかなるリンク先サイトの内容についても責任を持ちません。本マニュアルに記載されているサードパーティの Web サイトには、自己責任においてアクセスしてください。アドビ システムズ社はこれらのリンクを便宜上の目的においてのみ掲載しています。リンクを掲載することにより、アドビ システムズ社がこれらのサードパーティのサイトの内容について何らかの責任を持つことを示すものではありません。

 Sorenson™ Spark™ ビデオ圧縮および圧縮解除テクノロジーは、Sorenson Media, Inc. のライセンス供与によって提供されます。

Fraunhofer-IIS/Thomson Multimedia : MPEG レイヤー 3 音声圧縮テクノロジーは、Fraunhofer IIS および Thomson Multimedia (<http://www.iis.fhg.de/ammm/>) によりライセンス供与されています。

Independent JPEG Group : 本ソフトウェアの一部は、Independent JPEG Group による著作物に基づきます。

Nellymoser, Inc. : 音声圧縮および圧縮解除テクノロジーは、Nellymoser, Inc. (<http://www.nellymoser.com>) のライセンス供与によって提供されます。

Opera® browser Copyright © 1995-2002 Opera Software ASA and its suppliers. All rights reserved.

Macromedia Flash 8 ビデオは、On2 TrueMotion ビデオテクノロジーを利用しています。© 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Visual SourceSafe は、米国およびその他の国における Microsoft Corporation の登録商標または商標です。

更新情報およびその他のサードパーティのコード情報は、http://www.adobe.com/go/thirdparty_jp/ で入手できます。

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users.The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein.Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA.For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250 ,and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

目 次

第1章：ActionScript 言語エレメント	7
コンパイラディレクティブ	7
定数	11
グローバル関数	15
グローバルプロパティ	72
演算子	91
ステートメント	150
fscommand2 コマンド	194
 第2章：ActionScript クラス	 213
arguments	213
Array	215
Boolean	235
Button	237
capabilities (System.capabilities)	262
Color	281
Date	286
Error	316
ExtendedKey	320
Function	324
Key	328
LoadVars	346
Math	361
Mouse	378
MovieClip	385
MovieClipLoader	469
Number	485
Object	490
security (System.security)	508
Selection	514
SharedObject	522
Sound	534
Stage	556
String	563

System	578
TextField	581
TextFormat	627
Video	642
XML	647
XmlNode	668
XMLSocket	686
第 3 章：使用されなくなった ActionScript	695
使用されなくなった関数の一覧	695
使用されなくなったプロパティの一覧	697
使用されなくなった演算子の一覧	699
第 4 章：サポートされていない ActionScript	701
サポートされていないクラス	701
サポートされていないメソッド	701
サポートされていないプロパティ	702
サポートされていないグローバル関数	702
サポートされていないイベントハンドラ	702
サポートされていない fscommand	702
索引	703

第1章

ActionScript 言語エレメント

このセクションでは、グローバル関数とグローバルプロパティ（これらの言語エレメントは、ActionScript クラスには属していません）、コンパイラディレクティブ、および ActionScript で使用され ECMAScript (ECMA-262) Edition 4 の言語仕様草案で定義されている定数、演算子、ステートメント、およびキーワードに関して、シンタックス、使用法、およびコードサンプルをそれぞれ示します。

コンパイラディレクティブ

このセクションでは、コンパイラで特定の命令を前処理するために ActionScript ファイルで使用するディレクティブについて説明します。

コンパイラディレクティブ一覧

ディレクティブ	説明
#endinitclip	コンパイラディレクティブ:初期化アクションのブロックの終わりを示します。
#include	コンパイラディレクティブ:指定したファイル内のコマンドを呼び出し元のスクリプトにインクルードし、そのスクリプトの一部であるかのように扱います。
#initclip	コンパイラディレクティブ:初期化アクションのブロックの始まりを示します。

#endinitclip ディレクティブ

#endinitclip

コンパイラディレクティブ: 初期化アクションのブロックの終わりを示します。

例

```
#initclip  
...initialization actions go here...  
#endinitclip
```

#include ディレクティブ

`#include "[path]filename.as"` – #include ステートメントを含む行の最後にはセミコロン(;)を記述しません。

コンパイラディレクティブ: 指定したファイル内のコマンドを呼び出し元のスクリプトにインクルードし、そのスクリプトの一部であるかのように扱います。#include ディレクティブは、コンパイル時に起動されます。このため、外部ファイルに何らかの変更を行った場合には、ファイルを保存し、その外部ファイルを使用している FLA ファイルを再コンパイルする必要があります。

#include ステートメントを含むスクリプトに対して [シンタックスチェック] ボタンを使用すると、インクルードされるファイルのシンタックスもチェックされます。

#include アクションは、FLA ファイルおよび外部スクリプトファイルで使用できますが、ActionScript 2.0 クラスファイルでは使用できません。

インクルードするファイルのパスには相対パスも絶対パスも指定できますが、パスを省略することもできます。パスを省略する場合は、AS ファイルが次のいずれかの場所に存在する必要があります。

- FLA ファイルと同じディレクトリ。#include ステートメントを含むスクリプトと同じディレクトリ。
- 次のいずれかのグローバル Include ディレクトリ。
 - Windows 2000 または Windows XP : C:\Documents and Settings\ユーザー\Local Settings\Application Data\Macromedia\Flash 8\言語\Configuration\Include
 - Macintosh OS X : Hard Drive/Users/Library/Application Support/Macromedia/Flash 8/\言語\Configuration/Include
- <Flash 8 プログラム>\<言語>\First Run\Include ディレクトリ。このディレクトリにファイルを保存した場合、Flash を次回起動したときに、ファイルはグローバル Include ディレクトリにコピーされます。

AS ファイルの相対パスを指定するには、現在のディレクトリを表す 1 つのドット(.)、親ディレクトリを表す 2 つのドット(..)、およびサブディレクトリを表すスラッシュ(/)を使用します。次の例を参照してください。

AS ファイルの絶対パスを指定するには、そのプラットフォーム (Windows または Macintosh) でサポートされている形式を使用します。次の例を参照してください。ただし、この方法では、スクリプトをコンパイルするのに使用するコンピュータで同じディレクトリ構造を維持する必要があるので、推奨されません。

ファイルを "First Run/Include" ディレクトリまたはグローバル "Include" ディレクトリに保存する場合は、これらのファイルをバックアップしてください。Flash をアンインストールした後で再びインストールする必要性が生じた場合、これらのディレクトリは削除された後に上書きされてしまいます。

パラメータ

[path]filename.as - filename.as[アクション]パネルまたは現在のスクリプトに追加するスクリプトのファイル名とパス(省略可能)。推奨されるファイル拡張子は .as です。

例

次の例は、スクリプトにインクルードするファイルのパスを指定するさまざまな方法を示しています。

```
// Note that #include statements do not end with a semicolon (;)
// AS file is in same directory as FLA file or script
// or is in the global Include directory or the First Run/Include directory
#include "init_script.as"

// AS file is in a subdirectory of one of the above directories
// The subdirectory is named "FLA_includes"
#include "FLA_includes/init_script.as"
// AS file is in a subdirectory of the script file directory
// The subdirectory is named "SCRIPT_includes"
#include "SCRIPT_includes/init_script.as"
// AS file is in a directory at the same level as one of the above directories
// AS file is in a directory at the same level as the directory
// that contains the script file
// The directory is named "ALL_includes"
#include "../ALL_includes/init_script.as"

// AS file is specified by an absolute path in Windows
// Note use of forward slashes, not backslashes
#include "C:/Flash_scripts/init_script.as"

// AS file is specified by an absolute path on Macintosh
#include "Mac HD:Flash_scripts:init_script.as"
```

#initclip ディレクティブ

#initclip order - #initclip ステートメントを含む行の最後にはセミコロン(;)を記述しません。

コンパイラディレクティブ: 初期化アクションのブロックの始まりを示します。複数のクリップを同時に初期化する場合は、最初に初期化するクリップを *order* パラメータによって指定できます。ムービークリップシンボルを定義すると、初期化アクションが実行されます。ムービークリップが書き出されたシンボルである場合は、SWF ファイルのフレーム 1 にあるアクションの前に初期化アクションが実行されます。それ以外の場合は、関連するムービークリップシンボルの最初のインスタンスを含むフレームのフレームアクションの直前に実行されます。

初期化アクションは、SWF ファイルの再生時に 1 回だけ実行されます。初期化アクションは、クラス定義や登録など、1 回だけ実行する初期化処理に使用してください。

パラメータ

`order -#initclip` コードのブロックの実行順を指定する負以外の整数。このパラメータはオプションです。指定する場合は変数ではなく整数リテラルを使用し、値は 16 進法ではなく 10 進法で表す必要があります。1つのムービークリップシンボルに複数の `#initclip` ブロックが含まれる場合、そのムービークリップシンボルで指定された最後の `order` 値が、シンボル内のすべての `#initclip` ブロックに使用されます。

例

次の例では、ムービークリップインスタンス内のフレーム 1 に ActionScript を配置します。`"variables.txt"` テキストファイルは同じディレクトリにあるものとします。

```
#initclip

trace("initializing app");

var variables:LoadVars = new LoadVars();

variables.load("variables.txt");

variables.onLoad = function(success:Boolean) {

    trace("variables loaded:"+success);

    if (success) {
        for (i in variables) {
            trace("variables."+i+" = "+variables[i]);
        }
    }
};

#endinitclip
```

定数

定数とは、値が変化しないプロパティを表すのに使用する変数のことです。このセクションでは、どのスクリプトでも使用できるグローバル定数について説明します。

定数一覧

オプション	定数	説明
	<code>false</code>	<code>true</code> の逆を表す一意のブール値です。
	<code>Infinity</code>	正の無限大を表す IEEE-754 値を指定します。
	<code>-Infinity</code>	負の無限大を表す IEEE-754 値を指定します。
	<code>NaN</code>	<code>NaN</code> (非数) を表す IEEE-754 値を保持する定義済みの変数です。
	<code>newline</code>	キャリッジリターン文字 (\r) を挿入します。これにより、コードが生成するテキスト出力に空白行ができます。
	<code>null</code>	変数に代入できる特別な値、またはデータがない場合に関数から返される特別な値です。
	<code>true</code>	<code>false</code> の逆を表す一意のブール値です。
	<code>undefined</code>	通常、変数に値がまだ割り当てられていないことを示すために使用される特別な値です。

false 定数

`true` の逆を表す一意のブール値です。

自動的な型指定により `false` を数値に変換すると、その結果は `0` となります。`false` をストリングに変換すると、その結果は `"false"` となります。

例

この例では、自動的な型指定によって `false` がどのような数値やストリングに変換されるかを示します。

```
var booll:Boolean = Boolean(false);

// converts it to the number 0
trace(1 + booll); // outputs 1

// converts it to a string
trace("String: " + booll); // outputs String: false
```

Infinity 定数

正の無限大を表す IEEE-754 値を指定します。この定数の値は、Number.POSITIVE_INFINITY と同じです。

関連項目

[POSITIVE_INFINITY \(Number.POSITIVE_INFINITY プロパティ\)](#)

-Infinity 定数

負の無限大を表す IEEE-754 値を指定します。この定数の値は、Number.NEGATIVE_INFINITY と同じです。

関連項目

[NEGATIVE_INFINITY \(Number.NEGATIVE_INFINITY プロパティ\)](#)

NaN 定数

NaN(非数) を表す IEEE-754 値を保持する定義済みの変数です。数値が NaN かどうかを判別するには、[isNaN\(\)](#) を使用します。

関連項目

[isNaN 関数, NaN \(Number.NaN プロパティ\)](#)

newline 定数

キャリッジターン文字 (¥r) を挿入します。これにより、コードが生成するテキスト出力に空白行ができます。コード内の関数またはステートメントで検索される情報のためのスペースを作成するには、[newline](#) を使用します。

例

次の例では、[newline](#) を使って、[trace\(\)](#) ステートメントの出力を複数行に表示しています。

```
var myName:String = "Lisa", myAge:Number = 30;
trace(myName+myAge);
trace("-----");
trace(myName+newline+myAge);
// output:
Lisa30
-----
Lisa
30
```

関連項目

[trace 関数](#)

null 定数

変数に代入できる特別な値、またはデータがない場合に関数から返される特別な値です。null は、存在しない、または定義されていないデータ型を表す値として使用されます。

例

数値のコンテキストでは、null は 0 と評価されます。null を使用して等価テストを実行することができます。このステートメントでは、バイナリツリーノードに子がないので、子のフィールドを null に設定することができます。

```
if (tree.left == null) {  
    tree.left = new TreeNode();  
}
```

true 定数

false の逆を表す一意の布尔値です。自動的な型指定により true を数値に変換すると、その結果は 1 となります。true をストリングに変換すると、その結果は "true" となります。

例

次の例では、if ステートメントで true を使用しています。

```
var shouldExecute:Boolean;  
// ...  
// code that sets shouldExecute to either true or false goes here  
// shouldExecute is set to true for this example:  
  
shouldExecute = true;  
  
if (shouldExecute == true) {  
    trace("your statements here");  
}  
  
// true is also implied, so the if statement could also be written:  
// if (shouldExecute) {  
//     trace("your statements here");  
// }
```

次の例では、自動的な型指定によって true を数値 1 に変換する方法を示しています。

```
var myNum:Number;  
myNum = 1 + true;  
trace(myNum); // output: 2
```

関連項目

[false 定数](#), [Boolean](#)

undefined 定数

通常、変数に値がまだ割り当てられていないことを示すために使用される特別な値です。未定義の値を参照すると、特別な値 undefined が返されます。ActionScript コード `typeof(undefined)` は、ストリング "undefined" を返します。タイプ undefined の唯一の値は undefined です。

Flash Player 6 以前用にパブリッシュされたファイルでは、`String(undefined)` は ""(空のストリング) になります。Flash Player 7 以降用にパブリッシュされたファイルでは、`String(undefined)` は "undefined" になります (`undefined` がストリングに変換されます)。

Flash Player 6 以前用にパブリッシュされたファイルでは、`Number(undefined)` は 0 になります。Flash Player 7 以降用にパブリッシュされたファイルでは、`Number(undefined)` は NaN になります。値 undefined は特別な値 null に似ています。`null` と `undefined` を等価演算子 (==) で比較すると、結果は `true` になります。ただし、`null` と `undefined` を厳密な等価演算子 (===) で比較すると、結果は `false` になります。

例

次の例では、変数 `x` が宣言されていないので、値は `undefined` になります。

コードの第1セクションでは、等価演算子 (==) で `x` の値を `undefined` と比較し、その結果を [出力] パネルに表示しています。コードの第1セクションでは、等価演算子 (==) で `x` の値を `undefined` と比較し、その結果をログファイルに送信しています。

コードの第2セクションでは、等価 (==) 演算子で `null` 値と `undefined` 値を比較しています。

```
// x has not been declared
trace("The value of x is "+x);

if (x == undefined) {
    trace("x is undefined");
} else {
    trace("x is not undefined");
}

trace("typeof (x) is "+typeof (x));

if (null == undefined) {
    trace("null and undefined are equal");
} else {
    trace("null and undefined are not equal");
}
```

次の結果が [出力] パネルに表示されます。

```
The value of x is undefined  
x is undefined  
typeof (x) is undefined  
null and undefined are equal
```

グローバル関数

ここでは、ActionScript を使用している SWF ファイルで使用できるビルトイン関数について説明します。グローバル関数では、データ型の処理(Boolean()、int() など)、デバッグ情報の作成(trace())、Flash Player やブラウザとの通信(fscommand())など、各種の一般的なプログラミングタスクがカバーされています。

グローバル関数一覧

オプション	シグネチャ	説明
	<code>Array([numElements], [elementN]) : Array</code>	新しい空の配列を作成します。また、指定されたエレメントを配列に変換します。
	<code>Boolean(expression: Object) : Boolean</code>	<i>expression</i> パラメータをブール値に変換し、true または false を返します。
	<code>call(frame:Object)</code>	非推奨 Flash Player 5 使用しないでください。このアクションの代わりに function ステートメントを使用します。呼び出されたフレームで、そのフレームに再生ヘッドを移動せずに、スクリプトを実行します。
	<code>chr(number:Number) : String</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.fromCharCode() を使用します。ASCII コード番号を文字に変換します。
	<code>clearInterval(intervalID:Number)</code>	<code>setInterval()</code> の呼び出しによって作成された間隔をキャンセルします。
	<code>duplicateMovieClip(target:Object, newname:String, depth:Number)</code>	SWF ファイルの再生中にムービークリップのインスタンスを作成します。
	<code>escape(expression: String) : String</code>	パラメータをストリングに変換し、URL エンコードします。この場合、英数字以外のすべての文字は % が付いた 16 進シーケンスで置き換えられます。
	<code>eval(expression: Object) : Object</code>	変数、プロパティ、オブジェクト、ムービークリップに名前でアクセスします。

オプション	シグネチャ	説明
	<code>fscommand(command: String, parameters:String)</code>	SWF ファイルが Flash Lite プレーヤー、またはモバイルデバイスの環境(オペレーティングシステムなど)と通信できるようにします。
	<code>fscommand2 (command:String, parameters:String)</code>	SWF ファイルが Flash Lite プレーヤーまたはモバイルデバイス上のホストアプリケーションと通信できるようにします。
	<code>getProperty (my_mc:Object, property:Object) : Object</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに Flash Player 5 で実装されたドットシンタックスを使用します。 ムービークリップ <code>my_mc</code> の指定プロパティの値を返します。
	<code>getTimer() : Number</code>	SWF ファイルを再生し始めてからの経過時間をミリ秒単位で返します。
	<code>getURL(url:String, [window:String], [method:String])</code>	特定の URL からウィンドウにドキュメントをロードしたり、定義済みの URL に存在する別のアプリケーションに変数を渡したりします。
	<code>getVersion() : String</code>	Flash Player のバージョンとプラットフォーム情報を含むストリングを返します。
	<code>gotoAndPlay([scene: String], frame:Object)</code>	シーン内の指定されたフレームに再生ヘッドを送り、そのフレームから再生します。
	<code>gotoAndStop([scene:String], frame:Object)</code>	シーン内の指定されたフレームに再生ヘッドを送り、停止します。
	<code>ifFrameLoaded ([scene:String], frame:Object, statement(s):Object)</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数は使用されなくなりました。 <code>MovieClip._framesloaded</code> プロパティを使用することをお勧めします。 特定のフレームの内容がローカルに使用できるかどうかを確認します。
	<code>int(value:Number) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>Math.round()</code> を使用します。 小数値を切り捨てるによって、10 進数の整数部を取り出します。したがって、負数の扱いは <code>Math.floor()</code> × ソッドと異なります。
	<code>isFinite(expression: Object) : Boolean</code>	<code>expression</code> を評価し、有限大である場合は <code>true</code> を、無限大または負の無限大である場合は <code>false</code> を返します。

オプション	シグネチャ	説明
	<code>isNaN(expression: Object) : Boolean</code>	パラメータを評価し、値が NaN (非数) である場合は true を返します。
	<code>length(expression: String, variable:Object) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数およびすべてのストリング関数は使用されなくなりました。String クラスのメソッドと String.length プロパティを使用して同じ処理を行うことをお勧めします。指定されたストリングまたは変数の長さを返します。
	<code>loadMovie(url:String, target:Object, [method:String])</code>	元の SWF ファイルの再生中に SWF ファイルまたは JPEG ファイルを Flash Player 内にロードします。
	<code>loadMovieNum (url:String, level:Number, [method:String])</code>	ロードした元の SWF ファイルの再生中に SWF ファイルまたは JPEG ファイルを Flash Player のレベル内にロードします。
	<code>loadVariables (url:String, target:Object, [method:String])</code>	テキストファイルや、ColdFusion、CGI スクリプト、ASP (Active Server Pages)、パーソナルホームページ (PHP)、または Perl スクリプトで作成されたテキストなどの外部ファイルからデータを読み取り、ターゲットムービークリップの変数に値を設定します。
	<code>loadVariablesNum (url:String, level:Number, [method:String])</code>	テキストファイルや、ColdFusion、CGI スクリプト、ASP、PHP、または Perl スクリプトで作成されたテキストなどの外部ファイルからデータを読み取り、Flash Player の特定のレベルの変数に値を設定します。
	<code>mbchr(number:Number)</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.fromCharCode() メソッドを使用します。 ASCII コード番号をマルチバイト文字に変換します。
	<code>mblength(string: String) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.length プロパティを使用します。マルチバイト文字のストリング長を返します。
	<code>mbord(character: String) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.charCodeAt() メソッドを使用します。 指定された文字をマルチバイト文字コード番号に変換します。

オプション	シグネチャ	説明
	<code>mbsubstring (value:String, index:Number, count:Number) : String</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.substr()</code> メソッドを使用します。 マルチバイト文字ストリングから、新たにマルチバイト文字ストリングを抽出します。
	<code>nextFrame()</code>	次のフレームに再生ヘッドを送ります。
	<code>nextScene()</code>	次のシーンのフレーム 1 に再生ヘッドを送ります。
	<code>Number(expression: Object) : Number</code>	<i>expression</i> パラメータを数値に変換します。
	<code>Object([value: Object]) : Object</code>	空のオブジェクトを新規作成するか、指定された数値、ストリング、またはブール値をオブジェクトに変換します。
	<code>on(mouseEvent:Object)</code>	アクションをトリガするマウスイベントまたはキー押下を指定します。
	<code>onClipEvent (movieEvent:Object)</code>	ムービークリップの特定のインスタンスに対して定義されたアクションを起動します。
	<code>ord(character: String) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String</code> クラスのメソッドとプロパティを使用します。 文字を ASCII コード番号に変換します。
	<code>parseFloat(string: String) : Number</code>	ストリングを浮動小数に変換します。
	<code>parseInt(expression: String, [radix:Number]) : Number</code>	ストリングを整数に変換します。
	<code>play()</code>	タイムライン内で再生ヘッドを前へ進めます。
	<code>prevFrame()</code>	前のフレームに再生ヘッドを送ります。
	<code>prevScene()</code>	前のシーンのフレーム 1 に再生ヘッドを送ります。
	<code>random(value: Number) : Number</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>Math.random()</code> を使用します。 0 から <i>value</i> パラメータで指定された整数まで(この整数は含まない)の間のランダムな整数を返します。
	<code>removeMovieClip (target:Object)</code>	指定されたムービークリップを削除します。

オプション	シグネチャ	説明
	<code>setInterval (functionName: Object, interval:Number, [param:Object], objectName:Object, methodName:String) : Number</code>	SWF ファイルの再生時に一定の間隔で関数、メソッド、またはオブジェクトを呼び出します。
	<code>setProperty(target: Object, property:Object, expression:Object)</code>	ムービークリップの再生時にムービークリップのプロパティ値を変更します。
	<code>startDrag(target: Object, [lock:Boolean], [left,top,right, bottom:Number])</code>	ムービーの再生中に <i>target</i> ムービークリップをドラッグ可能にします。
	<code>stop()</code>	再生中の SWF ファイルを停止します。
	<code>stopAllSounds()</code>	再生ヘッドを停止せずに、SWF ファイルで再生中のサウンドをすべて停止します。
	<code>stopDrag()</code>	現在実行中のドラッグ操作を停止します。
	<code>String(expression: Object) : String</code>	指定されたパラメータのストリング表現を返します。
	<code>substring(string: String, index:Number, count:Number) : String</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.substr()</code> を使用します。ストリングの一部を抽出します。
	<code>targetPath (targetObject: Object) : String</code>	<i>movieClipObject</i> のターゲットパスをストリングとして返します。
	<code>tellTarget(target: String, statement(s):Object)</code>	非推奨 Flash Player 5 以降では使用しないでください。代わりにドット(.)表記と <code>with</code> ステートメントを使用することをお勧めします。 <i>statements</i> パラメータで指定された指示を、 <i>target</i> パラメータで指定されたタイムラインに適用します。

オプション	シグネチャ	説明
	<code>toggleHighQuality()</code>	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>_quality</code> を使用します。 Flash Player でアンチエイリアス処理のオンとオフを切り替えます。
	<code>trace(expression: Object)</code>	式を評価し、結果を出力します。
	<code>unescape(string: String) : String</code>	パラメータ <code>x</code> をストリングとして評価し、URL エンコードされた形式からストリングをデコード（すべての 16 進シーケンスを ASCII 文字に変換）して、ストリングを返します。
	<code>unloadMovie(target)</code>	<code>loadMovie()</code> を使ってロードしたムービークリップを Flash Player から削除します。
	<code>unloadMovieNum(level:Number)</code>	<code>loadMovieNum()</code> を使ってロードした SWF ファイルやイメージを Flash Player から削除します。

Array 関数

`Array(): Array` `Array(numElements:Number): Array` `Array([element0:Object [, element1, element2, ...elementN]]) : Array`

長さが 0 以上の新しい配列、または指定したエレメントのリスト（各種データ型など）により格納される配列を作成します。

次のいずれかを作成することができます。

- 空の配列
- 長さのみが指定されておりエレメント値が定義されていない配列
- エレメントが特定の値を持つ配列

この関数は、`Array` コンストラクタを使用して配列を作成するのに似ています。`Array` クラスのコンストラクタを参照してください。

エレメントの数 (`numElements`)、または 1 つ以上の異なる種類を含むエレメントリスト (`element0, element1, ... elementN`) を渡すことができます。

複数のデータ型を指定できるパラメータは、シグネチャで `Object` としてリストされます。

パラメータ

`numElements`(オプション) - 配列内のエレメント数を指定する正の整数。`numElements` またはエレメントリストのいずれかを指定できます。両方を指定することはできません。

`elementN`(オプション) - 1つ以上のパラメータ(`element0`, `element1`, ... , `elementN`)。任意のデータ型の値を指定できます。複数のデータ型を指定できるパラメータは、`Object`としてリストされます。`numElements` またはエレメントリストのいずれかを指定できます。両方を指定することはできません。

戻り値

`Array` - 配列。

例

```
var myArray:Array = Array();
myArray.push(12);
trace(myArray); //traces 12
myArray[4] = 7;
trace(myArray); //traces 12,undefined,undefined,7
```

シンタックス 2: 次の例では、エレメントを定義せずに、長さ 4 の配列を作成します。

```
var myArray:Array = Array(4);
trace(myArray.length); // traces 4
trace(myArray); // traces undefined,undefined,undefined,undefined
```

シンタックス 3: 次の例では、3つのエレメントが定義された配列を作成します。

```
var myArray:Array = Array("firstElement", "secondElement", "thirdElement");
trace (myArray); // traces firstElement,secondElement,thirdElement
Unlike the Array class constructor, the Array() function does not use the keyword new .
```

関連項目

[Array](#)

Boolean 関数

`Boolean(expression:Object) : Boolean`

次に示すように、パラメータ *expression* をブール値に変換して、値を返します。

- *expression* がブール値である場合、戻り値は *expression* です。
- *expression* が数値であれば、0 でない場合の戻り値は `true` となり、0 の場合は `false` となります。

expression がストリングである場合は、戻り値は次のようにになります。

- Flash Player 6 以前用にパブリッシュされたファイルでは、ストリングはまず数値に変換されます。この数値が0でない場合の戻り値は `true` となり、0 の場合は `false` となります。
- Flash Player 7 以降用にファイルをパブリッシュした場合の結果は、ストリングの長さがゼロより長ければ `true`、空のストリングであれば `false` になります。

expression がストリングである場合の結果は、ストリングの長さがゼロより長ければ `true`、空のストリングであれば `false` になります。

- *expression* が `undefined` または `NaN` (非数) であれば、戻り値は `false` です。
- *expression* がムービークリップまたはオブジェクトであれば、戻り値は `true` です。

`Boolean` クラスのコンストラクタとは異なり、`Boolean()` 関数はキーワード `new` を使用しません。さらに、`Boolean` クラスのコンストラクタは、パラメータが指定されない場合に `Boolean` オブジェクトを `false` に初期化しますが、`Boolean()` 関数は、パラメータが指定されない場合に `undefined` を返します。

パラメータ

`expression:Object` - ブール値に変換される式。

戻り値

`Boolean` - ブール値。

例

```
trace(Boolean(-1)); // output: true  
trace(Boolean(0)); // output: false  
trace(Boolean(1)); // output: true
```

```
trace(Boolean(true)); // output: true  
trace(Boolean(false)); // output: false
```

```
trace(Boolean("true")); // output: true  
trace(Boolean("false")); // output: true
```

```
trace(Boolean("Craiggers")); // output: true  
trace(Boolean ""); // output: false
```

ファイルを Flash Player 6 以前用にパブリッシュした場合、上記の 3 つの例は次のように異なる結果となります。

```
trace(Boolean("true")); // output: false  
trace(Boolean("false")); // output: false  
trace(Boolean("Craiggers")); // output: false
```

この例では、`Boolean()` 関数と `Boolean` クラスの使い方の大きな違いを示します。`Boolean()` 関数はブール値を作成し、`Boolean` クラスは `Boolean` オブジェクトを作成します。ブール値は値で比較され、`Boolean` オブジェクトは参照で比較されます。

```
// Variables representing Boolean values are compared by value  
var a:Boolean = Boolean("a"); // a is true  
var b:Boolean = Boolean(1); // b is true  
trace(a==b); // true
```

```
// Variables representing Boolean objects are compared by reference  
var a:Boolean = new Boolean("a"); // a is true  
var b:Boolean = new Boolean(1); // b is true  
trace(a == b); // false
```

関連項目

[Boolean](#)

call 関数

`call(frame)`

非推奨 Flash Player 5 以降では使用しないでください。このアクションの代わりに `function` ステートメントを使用します。

呼び出されたフレームで、そのフレームに再生ヘッドを移動せずに、スクリプトを実行します。スクリプトの実行後、ローカル変数は存在しません。

- ブロック内({})で変数を宣言しないで、アクションリストを `call()` アクションから実行した場合、変数はローカルであり、現在のリストの終わりで終了します。
- ブロック内で変数を宣言しないで、現在のアクションリストを `call()` アクションから実行しない場合、変数はタイムライン変数として解釈されます。

パラメータ

`frame:Object` - タイムラインのフレームのラベルまたは番号。

関連項目

[function ステートメント ,call \(Function.call メソッド \)](#)

chr 関数

`chr(number) : String`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `String.fromCharCode()` を使用します。

ASCII コード番号を文字に変換します。

パラメータ

`number:Number` - ASCII コード番号。

戻り値

`String` - 指定した ASCII コードの文字値。

例

`myVar = chr(65);` では、番号 65 を文字 A に変換し、それを変数 myVar に代入します。

関連項目

[fromCharCode \(String.fromCharCode メソッド\)](#)

clearInterval 関数

`clearInterval(intervalID:Number) : Void`

`setInterval()` の呼び出しによって作成された間隔をキャンセルします。

パラメータ

`intervalID:Number` - `setInterval()` の呼び出しから返される数値(整数)の識別子。

例

次の例では、最初に間隔への呼び出しを設定し、次にその呼び出しをクリアします。

```
function callback() {  
    trace("interval called: "+getTimer()+" ms.");  
}  
  
var intervalID:Number = setInterval(callback, 1000);
```

関数の使用を終了したときは間隔をクリアする必要があります。clearInt_btn という名前のボタンを作成し、次の ActionScript を使用して setInterval() をクリアします。

```
clearInt_btn.onRelease = function(){
    clearInterval( intervalID );
    trace("cleared interval");
};
```

関連項目

[setInterval 関数](#)

duplicateMovieClip 関数

```
duplicateMovieClip(target:String, newname:String, depth:Number) :  
Void  
duplicateMovieClip(target:MovieClip, newname:String, depth:Number) :  
Void
```

SWF ファイルの再生中にムービークリップのインスタンスを作成します。複製ムービークリップの再生ヘッドは、元のムービークリップでの再生ヘッドの位置に関係なく、常にフレーム 1 から始まります。元のムービークリップ内の変数は、複製されたムービークリップにコピーされません。

duplicateMovieClip() で作成されたムービークリップインスタンスを削除するには、removeMovieClip() 関数またはメソッドを使用します。

パラメータ

target:Object - 複製するムービークリップのターゲットパス。このパラメータには、String ("my_mc" など)、またはムービークリップインスタンス (my_mc など) への直接参照のいずれかを指定できます。複数のデータ型を指定できるパラメータは、Object としてリストされます。

newname:String - 複製したムービークリップのインスタンス名。

depth:Number - 複製したムービークリップ固有の深度。深度は、複製したムービークリップの重ね順です。この重ね順は、タイムラインの重ね順に似ています。深度の低いムービークリップは、より高い重ね順のクリップの下に隠されます。既に占有されている深度の SWF ファイルが置き換えられるのを避けるため、複製したムービークリップには、それぞれ固有の深度を割り当てます。

例

次の例では、新しいムービークリップインスタンス img_mc を作成します。イメージをムービークリップにロードした後、img_mc クリップを複製します。複製されたクリップの名前は newImg_mc です。この新しいクリップを元のクリップと重ならないようにステージに移動し、同じイメージを 2 番目のクリップにロードします。

```
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
img_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(img_mc, "newImg_mc", this.getNextHighestDepth());
```

```
newImg_mc._x = 200;
newImg_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
複製されたムービークリップを削除するには、ボタンmyButton_btnに次のコードを追加します。
this.myButton_btn.onRelease = function(){
    removeMovieClip(newImg_mc);
};
```

関連項目

[removeMovieClip 関数](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#), [removeMovieClip \(MovieClip.removeMovieClip メソッド\)](#)

escape 関数

`escape(expression:String) : String`

パラメータをストリングに変換し、URL エンコードします。この場合、英数字以外のすべての文字は % が付いた 16 進シケンスで置き換えられます。URL エンコードされたストリング内のパーセント記号 (%) は、エスケープ文字の開始を表すもので、剩余演算子 (%) ではありません。

パラメータ

`expression:String` - ストリングに変換し、URL エンコードする対象の式。

戻り値

`String` - URL エンコードされたストリング。

例

次のコードを作成すると、"someuser%40somedomain%2Ecom" が生成されます。

```
var email:String = "someuser@somedomain.com";
trace(escape(email));
```

この例では、アットマーク (@) が %40 に、ピリオド (.) が %2E に置き換えられます。この関数は、次のコードに示すように、リモートサーバーに送信する情報の中に特殊文字 (たとえば & や ?) が含まれている場合に便利です。

```
var redirectUrl = "http://www.somedomain.com?loggedin=true&username=Gus";
getURL("http://www.myothersite.com?returnurl="+ escape(redirectUrl));
```

関連項目

[unescape 関数](#)

eval 関数

`eval(expression: Object) : Object eval(expression:String) : Object`

変数、プロパティ、オブジェクト、ムービークリップに名前でアクセスします。`expression` が変数またはプロパティである場合は、変数またはプロパティの値が返されます。`expression` がオブジェクトまたはムービークリップである場合は、オブジェクトまたはムービークリップへの参照が返されます。`expression` に指定したエレメントが見つからない場合は、`undefined` が返されます。

Flash 4 では、配列をシミュレートするために `eval()` 関数を使用していましたが、Flash 5 以降では `Array` クラスを使用する必要があります。

Flash 4 では、`eval()` を使って変数の値またはインスタンスの名前を動的に設定および取得することもできます。ただし、同じことは配列アクセス演算子`([])`を使用しても可能です。

Flash 5 以降では、`eval()` を使用して変数の値またはインスタンス名を動的に設定および取得することはできません。これは、等式の左辺に `eval()` を使用できないためです。たとえば、次のようなコード

```
eval ("var" + i) = "first";
```

は、次のコードに置き換える必要があります。

```
this["var"+i] = "first"
```

または、次のようにもできます。

```
set ("var" + i, "first");
```

パラメータ

`expression:Object` - 取得する変数、プロパティ、オブジェクト、またはムービークリップの名前。このパラメータには、`String` またはオブジェクトインスタンスへの直接参照のいずれかを指定できます（引用符`("")`は使用しなくてもかまいません）。

戻り値

`Object` - オブジェクトまたはムービークリップへの参照、`undefined`、または値。

例

次の例では、`eval()` を使用して、動的に指定されるムービークリップのプロパティを設定します。この ActionScript は、`square1_mc`、`square2_mc`、`square3_mc` という 3 つのムービークリップの `_rotation` プロパティを設定します。

```
for (var i = 1; i <= 3; i++) {  
    setProperty(eval("square"+i+"_mc"), _rotation, 5);  
}
```

次の ActionScript を使用することもできます。

```
for (var i = 1; i <= 3; i++) {  
    this["square"+i+"_mc"]._rotation = -5;  
}
```

関連項目

[Array, set variable ステートメント](#)

fscommand 関数

`fscommand(command:String, parameters:String) : Void`

`fscommand()` 関数を使用して、SWF ファイルが Flash Lite プレーヤー、またはモバイルデバイスの環境（オペレーティングシステムなど）と通信できるようにします。パラメータは、開始されるアプリケーションの名前とそのパラメータをカンマで区切って定義します。

コマンド	パラメータ	用途
launch	application-path, arg1, arg2,..., argn	<p>このコマンドは、モバイルデバイス上の別のアプリケーションを起動します。起動されるアプリケーションの名前とそのパラメータが1つの引数として渡されます。</p> <p>メモ: この機能は、オペレーティングシステムに依存しています。このコマンドは、ホストデバイス上でサポートされていない操作を実行する可能性があるので、注意して使用してください。そのような使い方をすると、ホストデバイスがクラッシュする可能性があります。</p> <p>このコマンドは、Flash Lite プレーヤーがスタンダローンモードで実行されている場合にのみサポートされています。Flash Lite プレーヤーが他のアプリケーションのコンテキストで実行されている場合（ブラウザのプラグインとして実行されている場合など）、このコマンドは使用できません。</p>

コマンド	パラメータ	用途
activateTextField	"" (ignored)	<p>このコマンドは、現在選択されているテキストフィールドを非同期でアクティブにして、ユーザーが編集できるようにします。このコマンドは非同期で動作するため、フレームの最後で処理されます。fscommand()への呼び出しがまず実行されてから、すぐに ActionScript が処理されます。コマンドが処理されるときにテキストフィールドが選択されていない場合は、何も起こりません。</p> <p>このコマンドは、以前 Selection.setFocus() メソッドに渡されたテキストフィールドにフォーカスを与え、編集するためにテキストフィールドをアクティブにします。このコマンドは、ハンドセットがオンラインテキスト編集をサポートしている場合にのみ有効です。</p> <p>このコマンドは、Selection.onSetFocus() イベントリスナーコールバックの一部として呼び出すことができます。これにより、テキストフィールドが選択されると編集するためにアクティブになります。</p> <p>メモ: fscommand() 関数は非同期で実行されるため、テキストフィールドはすぐにはアクティブになりません。フレームの最後でアクティブになります。</p>

パラメータ

command:String - ホストアプリケーションに任意の用途で渡されるストリング、または Flash Lite プレーヤーに渡されるコマンド。

parameters:String - ホストアプリケーションに任意の用途で渡されるストリング、または Flash Lite プレーヤーに渡される値。

例

次の例では、fscommand() 関数を使用して、Series 60 携帯端末のサービス /Web ブラウザで wap.yahoo.com を開きます。

```
on(keyPress "9") {
    status = fscommand("launch",
        "z:¥¥system¥¥apps¥¥browser¥¥browser.app,http://wap.yahoo.com");
}
```

fscommand2 関数

`fscommand2(command:String, parameter1:String,...parameterN:String) : Void`

SWF ファイルが Flash Lite プレーヤーまたはモバイルデバイス上のホストアプリケーションと通信できるようにします。

`fscommand2()` を使用して Flash Lite プレーヤーにメッセージを送るには、定義済みのコマンドとパラメータを使用します。`fscommand()` 関数のコマンドおよびパラメータに指定できる値については、「ActionScript 言語エレメント」にある「[fscommand2 コマンド](#)」の項を参照してください。これらの値は、Flash Lite プレーヤーで再生する SWF ファイルを制御します。

`fscommand2()` 関数は `fscommand()` 関数と似ていますが、次のような違いがあります。

- `fscommand2()` 関数は、任意の数の引数を取れます。これに対し、`fscommand()` が取れる引数は1つだけです。
- `fscommand2()` はすぐに(つまりフレーム内で)実行されますが、`fscommand()` は処理中のフレームの最後に実行されます。
- `fscommand2()` 関数は、成功または失敗の報告に使用できる値、またはコマンドの結果を返します。

メモ: Web プレーヤーでは、`fscommand2()` コマンドを使用できません。

使用されなくなった fscommand2() コマンド

Flash Lite 1.1 の `fscommand2()` の中には、Flash Lite 2.0 で使用できなくなったものがあります。次の表に使用できなくなった `fscommand2()` コマンドを示します。

コマンド	代わりに使用するコマンド
Escape	<code>escape</code> グローバル関数
GetDateDay	Date オブジェクトの <code>getDate()</code> メソッド
GetDateMonth	Date オブジェクトの <code>getMonth()</code> メソッド
GetDateWeekday	Date オブジェクトの <code>getDay()</code> メソッド
GetDateYear	Date オブジェクトの <code>getYear()</code> メソッド
GetLanguage	<code>System.capabilities.language</code> プロパティ
GetLocaleLongDate	Date オブジェクトの <code>getLocaleLongDate()</code> メソッド

コマンド	代わりに使用するコマンド
GetLocaleShortDate	Date オブジェクトの getLocaleShortDate() メソッド
GetLocaleTime	Date オブジェクトの getLocaleTime() メソッド
GetTimeHours	Date オブジェクトの getHours() メソッド
GetTimeMinutes	Date オブジェクトの getMinutes() メソッド
GetTimeSeconds	Date オブジェクトの getSeconds() メソッド
GetTimeZoneOffset	Date オブジェクトの getTimeZoneOffset() メソッド
SetQuality	MovieClip._quality
Unescape	unescape() グローバル関数

パラメータ

command:String - ホストアプリケーションに任意の用途で渡されるストリング、または Flash Lite プレーヤーに渡されるコマンド。

parameters:String - ホストアプリケーションに任意の用途で渡されるストリング、または Flash Lite プレーヤーに渡される値。

getProperty 関数

`getProperty(my_mc:object, property:object) : Object`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに Flash Player 5 で実装されたドットシンタックスを使用します。

ムービークリップ *my_mc* の指定プロパティの値を返します。

パラメータ

my_mc:Object - プロパティを取得するムービークリップのインスタンス名。

property:Object - ムービークリップのプロパティ。

戻り値

Object - 指定したプロパティの値。

例

次の例では、新しいムービークリップ someClip_mc を作成し、ムービークリップ someClip_mc のアルファ値 (_alpha)を [出力] パネルに表示します。

```
this.createEmptyMovieClip("someClip_mc", 999);
trace("The alpha of "+getProperty(someClip_mc, _name)+" is:
      "+getProperty(someClip_mc, _alpha));
```

getTimer 関数

getTimer() : Number

SWF ファイルを再生し始めてからの経過時間をミリ秒単位で返します。

戻り値

Number - SWF ファイルの再生を開始してから経過したミリ秒数。

例

次の例では、*getTimer()* 関数と *setInterval()* 関数を使用して、単純なタイマーを作成します。

```
this.createTextField("timer_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
function updateTimer():Void {
  timer_txt.text = getTimer();
}

var intervalID:Number = setInterval(updateTimer, 100);
```

getURL 関数

getURL(url:String [, window:String [, method:String]]) : Void

特定の URL からウィンドウにドキュメントをロードしたり、定義済みの URL に存在する別のアプリケーションに変数を渡したりします。この関数をテストするには、ロードするファイルが指定した場所にあることを確認します。絶対 URL (<http://www.myserver.com>など) を使用するには、ネットワーク接続が確立されている必要があります。

メモ: この関数は、BREW デバイスではサポートされていません。

パラメータ

url:String - ドキュメントを取得するための URL。

window:String (オプション) - ドキュメントのロード先のウィンドウまたは HTML フレームを指定します。特定のウィンドウの名前を入力するか、次の予約されたターゲット名から選択します。

- *_self* は、現在のウィンドウ内の現在のフレームを指定します。
- *_blank* は、新規ウィンドウを指定します。

- `_parent` は、現在のフレームの親を指定します。
- `_top` は、現在のウィンドウ内の最上位のフレームを指定します。

`method:String(オプション)`- 変数を送るための GET メソッドまたは POST メソッド。変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、イメージをムービークリップにロードします。イメージがクリックされると、新しいブラウザウィンドウに新しい URL がロードされます。

```
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onRelease = function() {
    getURL("http://www.macromedia.com/software/flash/flashpro/", "_blank");
  };
};
var logo:MovieClipLoader = new MovieClipLoader();
logo.addListener(listenerObject);
logo.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  this.createEmptyMovieClip("macromedia_mc", this.getNextHighestDepth()));
```

次の例では、`getURL()` を使用して、電子メールメッセージを送信します。

```
myBtn_btn.onRelease = function(){
  getURL("mailto:you@somedomain.com");
};
```

GET や POST を使用して変数を送信することもできます。次の例では、GET を使用して、変数を URL に追加します。

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
myBtn_btn.onRelease = function() {
  getURL("http://www.macromedia.com", "_blank", "GET");
};
```

次の ActionScript では、POST を使用して、HTTP ヘッダ内で変数を送信します。ブラウザウィンドウでドキュメントをテストしてください。そうでないと、変数は GET を使って送信されます。

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
getURL("http://www.macromedia.com", "_blank", "POST");
```

関連項目

[loadVariables 関数](#), [send \(XML.send メソッド\)](#), [sendAndLoad \(XML.sendAndLoad メソッド\)](#)

getVersion 関数

`getVersion() : String`

Flash Player のバージョンとプラットフォーム情報を含むストリングを返します。`getVersion` 関数は、Flash Player 5 以降のバージョン情報のみを返します。

戻り値

`String` - Flash Player のバージョンとプラットフォーム情報を含むストリング。

例

次の例では、SWF ファイルを再生している Flash Player のバージョン番号をトレースします。

```
var flashVersion:String = getVersion();
trace(flashVersion); // output: WIN 8,0,1,0
trace($version); // output: WIN 8,0,1,0
trace(System.capabilities.version); // output: WIN 8,0,1,0
```

次のストリングが `getVersion` 関数から返されます。

`WIN 8,0,1,0`

この返されたストリングは、プラットフォームが Microsoft Windows であり、Flash Player がメジャーバージョン 8、マイナーバージョン 1(8.1)であることを示します。

関連項目

[os \(capabilities.os プロパティ\)](#), [version \(capabilities.version プロパティ\)](#)

gotoAndPlay 関数

`gotoAndPlay([scene:String,] frame:Object) : Void`

シーン内の指定されたフレームに再生ヘッドを送り、そのフレームから再生します。シーンを指定しないと、再生ヘッドは現在のシーン内の指定されたフレームに進みます。`scene` パラメータはルートタイムラインでのみ使用でき、ムービークリップやドキュメント内の他のオブジェクトのタイムラインでは使用できません。

パラメータ

`scene:String` (オプション) - 再生ヘッドの送り先となるシーンの名前を示すストリング。

`frame:Object` - 再生ヘッドの送り先となるフレーム番号を表す数値、または再生ヘッドの送り先となるフレームのラベルを表すストリング。

例

次の例では、`sceneOne` および `sceneTwo` の 2 つのシーンがドキュメントに含まれています。シーン 1 には、フレーム 10 の `newFrame` フレームラベルと、2 つのボタン `myBtn_btn` および `myOtherBtn_btn` が含まれています。次の ActionScript をメインタイムラインのシーン 1 のフレーム 1 に配置します。

```
stop();
myBtn_btn.onRelease = function(){
  gotoAndPlay("newFrame");
};

myOtherBtn_btn.onRelease = function(){
  gotoAndPlay("sceneTwo", 1);
};
```

ユーザーがボタンをクリックすると、指定された場所に再生ヘッドが移動した後、再生が続行されます。

関連項目

[gotoAndPlay](#) (`MovieClip.gotoAndPlay` メソッド), [nextFrame](#) 関数, [play](#) 関数, [prevFrame](#) 関数

gotoAndStop 関数

`gotoAndStop([scene:String,] frame:Object) : Void`

シーン内の指定されたフレームに再生ヘッドを送り、停止します。再生ヘッドは、シーンが指定されていない場合には現在のシーン内のフレームに送られます。`scene` パラメータはルートタイムラインでのみ使用でき、ムービークリップやドキュメント内の他のオブジェクトのタイムラインでは使用できません。

パラメータ

`scene:String` (オプション) - 再生ヘッドの送り先となるシーンの名前を示すストリング。

`frame:Object` - 再生ヘッドの送り先となるフレーム番号を表す数値、または再生ヘッドの送り先となるフレームのラベルを表すストリング。

例

次の例では、`sceneOne` および `sceneTwo` の 2 つのシーンがドキュメントに含まれています。シーン 1 には、フレーム 10 の `newFrame` フレームラベルと、2 つのボタン `myBtn_btn` および `myOtherBtn_btn` が含まれています。次の ActionScript をメインタイムラインのシーン 1 のフレーム 1 に配置します。

```
stop();  
  
myBtn_btn.onRelease = function(){  
    gotoAndStop("newFrame");  
};  
  
myOtherBtn_btn.onRelease = function(){  
    gotoAndStop("sceneTwo", 1);  
};
```

ユーザーがボタンをクリックすると、指定された場所に再生ヘッドが移動した後、停止します。

関連項目

[gotoAndStop \(MovieClip.gotoAndStop メソッド\)](#), [stop 関数](#), [play 関数](#), [gotoAndPlay 関数](#)

ifFrameLoaded 関数

```
ifFrameLoaded( [scene,] frame) { statement(s); }
```

非推奨 Flash Player 5 以降では使用しないでください。この関数は使用されなくなりました。

`MovieClip._framesloaded` プロパティを使用することをお勧めします。

特定のフレームの内容がローカルに使用できるかどうかを確認します。SWF ファイル全体がローカルコンピュータにダウンロードされるのを待つ間に単純なアニメーションの再生を開始する場合は、`ifFrameLoaded` を使用します。`_framesloaded` が `ifFrameLoaded` と異なる点は、`_framesloaded` では、独自の `if` ステートメントや `else` ステートメントを追加できることです。

パラメータ

`scene:String` (オプション) - ロードする必要のあるシーンの名前を示すストリング。

`frame:Object` - 次のステートメントを実行する前にロードするフレーム番号またはフレームラベル。

`statement(s):Object` - 指定されたフレーム、またはシーンおよびフレームのロードに応じて実行される指示。

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#)

int 関数

`int(value) : Number`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `Math.round()` を使用します。

小数値を切り捨てるによって、10 進数の整数部を取り出します。したがって、負数の扱いは `Math.floor()` メソッドと異なります。この関数は `Math.floor()` (`value` パラメータが正の場合) および `Math.ceil()` (`value` パラメータが負の場合) と同じです。

パラメータ

`value: Number` - 整数部を取り出す数値。

戻り値

`Number` - 取り出された整数値。

関連項目

[round \(Math.round メソッド\)](#), [floor \(Math.floor メソッド\)](#), [ceil \(Math.ceil メソッド\)](#)

isFinite 関数

`isFinite(expression:Object) : Boolean`

`expression` を評価し、有限大である場合は `true` を、無限大または負の無限大である場合は `false` を返します。無限大または負の無限大は、0 による除算などの数学的なエラーの可能性を示します。

パラメータ

`expression:Object` - 評価するブール値、変数、または式。

戻り値

`Boolean` - ブール値。

例

`isFinite` の戻り値の例を次に示します。

```
isFinite(56)
// returns true

isFinite(Number.POSITIVE_INFINITY)
//returns false
```

isNaN 関数

`isNaN(expression:Object) : Boolean`

パラメータを評価し、値が NaN (非数) である場合は true を返します。この関数は、式が正常に評価されて数値になるかどうかをチェックする場合に便利です。

パラメータ

`expression:Object` - 評価される布尔值、变数、または式。

戻り値

`Boolean` - ブール值。

例

次のコードは、`isNaN()` 関数の戻り値を示します。

```
trace( isNaN("Tree") );
// returns true
```

```
trace( isNaN(56) );
// returns false
```

```
trace( isNaN(Number.POSITIVE_INFINITY) )
// returns false
```

次の例では、`isNaN()` を使用して、式にエラーが含まれるかどうかをチェックする方法を示しています。

```
var dividend:Number;
var divisor:Number;
divisor = 1;
trace( isNaN(dividend/divisor) );
// output: true
// The output is true because the variable dividend is undefined.
// Do not use isNaN() to check for division by 0 because it will return false.
// A positive number divided by 0 equals Infinity (Number.POSITIVE_INFINITY).
// A negative number divided by 0 equals -Infinity (Number.NEGATIVE_INFINITY).
```

関連項目

[NaN 定数](#), [NaN \(Number.NaN プロパティ\)](#)

length 関数

`length(expression)length(variable)`

非推奨 Flash Player 5 以降では使用しないでください。この関数およびすべてのストリング関数は使用されなくなりました。`String` クラスのメソッドと `String.length` プロパティを使用して同じ処理を行うことをお勧めします。

指定されたストリングまたは変数の長さを返します。

パラメータ

`expression:String` - ストリング。

`variable:Object` - 変数の名前。

戻り値

`Number` - 指定したストリングまたは変数の長さ。

例

次の例では、ストリング "Hello" の長さを返します。`length("Hello")`; 結果は 5 です。

関連項目

"[ストリング区切り記号演算子](#), `String.length` (`String.length` プロパティ)

loadMovie 関数

`loadMovie(url:String, target:Object [, method:String]) : Void`
`loadMovie(url:String, target:String [, method:String]) : Void`

元の SWF ファイルの再生中に SWF ファイルまたは JPEG ファイルを Flash Player 内にロードします。プログレッシブ形式で保存されている JPEG ファイルはサポートされていません。

ヒント: ダウンロードの進捗状況を監視する場合は、この関数の代わりに `MovieClipLoader.loadClip()` を使用してください。

`loadMovie()` 関数を使用すると、複数の SWF ファイルを同時に表示し、別の HTML ドキュメントをロードせずに SWF ファイルを切り替えることができます。`loadMovie()` 関数を使用しないと、Flash Player は 1 つの SWF ファイルを表示します。

SWF ファイルまたは JPEG ファイルを特定のレベルにロードするには、`loadMovie()` の代わりに `loadMovieNum()` を使用します。

SWF ファイルをターゲットムービークリップにロードすると、そのムービークリップのターゲットパスを使用して、ロードした SWF ファイルをターゲットとすることができます。ターゲットにロードした SWF ファイルまたはイメージは、ターゲットムービークリップの位置、回転、および拡大 / 縮小プロパティを継承します。ロードしたイメージまたは SWF ファイルの左上隅は、ターゲットムービークリップの基準点に位置が合わされます。ターゲットがルートタイムラインである場合、イメージまたは SWF ファイルの左上隅はステージの左上隅に位置が合わされます。

`loadMovie()` を使ってロードした SWF ファイルを削除するには、`unloadMovie()` を使用します。

パラメータ

url:String - ロードする SWF ファイルまたは JPEG ファイルの絶対 URL または相対 URL。相対パスは、レベル 0 の SWF ファイルが埋め込まれた HTML ファイルを基準にする必要があります。絶対 URL の場合は `http://` や `file:///` などのプロトコル参照を含めて指定します。

target:Object - ターゲットムービークリップへのパスを表すムービークリップオブジェクトまたはストリングへの参照。ターゲットムービークリップは、ロードした SWF ファイルまたはイメージに置き換えられます。

method:String(オプション) - 変数を送信するための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送る変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

シンタックス 1: 次の例では、ステージに存在する `mySquare` というムービークリップを、同じディレクトリからロードした SWF ファイル "circle.swf" と置き換えます。

```
loadMovie("circle.swf", mySquare);
// equivalent statement (Usage 1): loadMovie("circle.swf", _level0.mySquare);
// equivalent statement (Usage 2): loadMovie("circle.swf", "mySquare");
```

次の例では、SWF ファイル "circle.swf" を同じディレクトリからロードしますが、`mySquare` ムービークリップではなくメインムービークリップを置き換えます。

```
loadMovie("circle.swf", this);
// Note that using "this" as a string for the target parameter will not work
// equivalent statement (Usage 2): loadMovie("circle.swf", "_level0");
```

次の `loadMovie()` ステートメントは、`createEmptyMovieClip()` を使って作成された新しいムービークリップ `logo_mc` に SWF ファイル "sub.swf" を同じディレクトリからロードします。

```
this.createEmptyMovieClip("logo_mc", 999);
loadMovie("sub.swf", logo_mc);
```

次のコードを追加すると、"sub.swf" をロードするときに同じディレクトリから JPEG イメージ "image1.jpg" を SWF ファイルにロードできます。JPEG イメージは、ボタン myBtn_btn をクリックするとロードされます。このコードは、JPEG イメージを logo_mc にロードします。したがって、"sub.swf" は JPEG イメージで置き換えられます。

```
myBtn_btn.onRelease = function(){
    loadMovie("image1.jpg", logo_mc);
};
```

シンタックス 2: 次の例では、ステージに存在する mySquare というムービークリップを、同じディレクトリからロードした SWF ファイル "circle.swf" と置き換えます。

```
loadMovie("circle.swf", "mySquare");
```

関連項目

[_level プロパティ](#), [loadMovieNum 関数](#), [loadMovie \(MovieClip.loadMovie メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [unloadMovie 関数](#)

loadMovieNum 関数

`loadMovieNum(url:String, level:Number [, method:String]) : Void`

ロードした元の SWF ファイルの再生中に SWF ファイルまたは JPEG ファイルを Flash Player のレベル内にロードします。

ヒント: ダウンロードの進捗状況を監視する場合は、この関数の代わりに MovieClipLoader.loadClip() を使用してください。

通常、Flash Player は 1 つの SWF ファイルを表示した後に閉じます。loadMovieNum() アクションを使用すると、複数の SWF ファイルを同時に表示し、別の HTML ドキュメントをロードせずに SWF ファイルを切り替えることができます。

レベルではなくターゲットを指定するには、loadMovieNum() の代わりに loadMovie() を使用します。Flash Player では、レベル 0 からレベルが積み重ねられます。これらのレベルは各レベルのオブジェクト以外は透明であり、フィルムのレイヤーに似ています。loadMovieNum() を使用する場合は、SWF ファイルをロードする先の Flash Player のレベルを指定します。SWF ファイルをレベル内にロードすると、シンタックス _level N を使用できます。N は SWF ファイルのターゲットとなるレベル番号です。

SWF ファイルをロードするときは、任意のレベル番号を指定できます。SWF ファイルが既にロードされているレベルに SWF ファイルをロードすることもできます。その場合は、新しい SWF ファイルによって既存の SWF ファイルが置き換えられます。SWF ファイルをレベル 0 内にロードすると、Flash Player の各レベルはアンロードされ、レベル 0 は新しいファイルに置き換えられます。レベル 0 の SWF ファイルによって、ロードした他のすべての SWF ファイルのフレームレート、背景色、およびフレームサイズが設定されます。

`loadMovieNum()` アクションを使用すると、JPEG ファイルを再生中の SWF にロードすることもできます。イメージと SWF ファイルのいずれの場合でも、ファイルのロード時にイメージの左上隅がステージの左上隅に位置合わせされます。また、どちらの場合でも、ロードしたファイルは回転と拡大 / 縮小を継承し、元の内容が指定されたレベルで上書きされます。

メモ : プログレッシブ形式で保存されている JPEG ファイルはサポートされていません。

`loadMovieNum()` を使ってロードしたムービーやイメージを削除するには、`unloadMovieNum()` を使用します。

パラメータ

`url:String` - ロードする SWF ファイルまたは JPEG ファイルの絶対 URL または相対 URL。相対パスの場合は、レベル 0 の SWF ファイルを基準にする必要があります。スタンドアローンの Flash Player 内で使用する際、または Flash オーサリングアプリケーション内のプレビューモードでテストする際には、すべての SWF ファイルを同じフォルダに置く必要があり、ファイル名にフォルダやディスクドライブの指定を含めることはできません。

`level:Number` - SWF ファイルをロードする先の Flash Player のレベルを指定する整数。

`method:String` (オプション) - 変数を送信するための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送る変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、JPEG イメージ "tim.jpg" を Flash Player のレベル 2 内にロードします。

```
loadMovieNum("http://www.helpexamples.com/flash/images/image1.jpg", 2);
```

関連項目

[unloadMovieNum 関数](#), [loadMovie 関数](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [_level プロパティ](#)

loadVariables 関数

`loadVariables(url:String, target:Object [, method:String]) : Void`

テキストファイルや、ColdFusion、CGI スクリプト、ASP (Active Server Pages)、パーソナルホームページ (PHP)、または Perl スクリプトで作成されたテキストなどの外部ファイルからデータを読み取り、ターゲットムービークリップの変数に値を設定します。このアクションを使用して、現在の SWF ファイルの変数を新しい値で更新することもできます。

指定された URL にあるテキストは、標準の MIME 形式 *application/x-www-form-urlencoded* (CGI スクリプトで使用される標準形式) である必要があります。変数はいくつでも指定できます。たとえば、次の例では複数の変数が定義されています。

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、*url* は、呼び出し元の SWF ファイルと同じスパーードメインに属している必要があります。スパーードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、www.someDomain.com に存在する SWF ファイルは、store.someDomain.com に存在するデータソースからデータをロードできます。これは、どちらのファイルも同じスパーードメイン someDomain.com に属しているからです。

Flash Player 7 以降で動作する SWF ファイルでは、*url* に、この呼び出しを発行する SWF ファイルと完全に同じドメインを使用する必要があります (『ActionScript ユーザーガイド』の「Flash Player のセキュリティ機能」を参照)。たとえば、www.someDomain.com に存在する SWF ファイルは、www.someDomain.com に存在するソースからしかデータをロードできません。異なるドメインからデータをロードする場合は、アクセスされるソースファイルをホスティングするサーバーに "クロスドメインポリシーファイル" を置いておく必要があります。詳細については、『ActionScript ユーザーガイド』の「ドメイン間のデータロード許可について」を参照してください。

変数を特定のレベルにロードするには、*loadVariables()* の代わりに *loadVariablesNum()* を使用します。

パラメータ

url:String - 変数が存在する絶対 URL または相対 URL。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、*url* は SWF ファイルと同じドメインに属している必要があります。詳細については、「説明」を参照してください。

target:Object - ロードした変数を受け取るムービークリップへのターゲットパス。

method:String (オプション) - 変数を送信するための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送る変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、テキストファイル "params.txt" の情報を、createEmptyMovieClip() を使用して作成されたムービークリップ target_mc にロードします。setInterval() 関数は、ロードの進捗状況をチェックする場合に使用します。このスクリプトは、"params.txt" ファイル内の done という名前の変数をチェックします。

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
loadVariables("params.txt", target_mc);
function checkParamsLoaded() {
  if (target_mc.done == undefined) {
    trace("not yet.");
  } else {
    trace("finished loading. killing interval.");
    trace("-----");
    for (i in target_mc) {
      trace(i+": "+target_mc[i]);
    }
    trace("-----");
    clearInterval(param_interval);
  }
}
var param_interval = setInterval(checkParamsLoaded, 100);
```

外部ファイル params.txt には、次のテキストが含まれています。

```
var1="hello"&var2="goodbye"&done="done"
```

関連項目

[loadVariablesNum 関数](#), [loadMovie 関数](#), [loadMovieNum 関数](#), [getURL 関数](#), [loadMovie\(MovieClip.loadMovie メソッド\)](#), [loadVariables \(MovieClip.loadVariables メソッド\)](#), [load \(LoadVars.load メソッド\)](#)

loadVariablesNum 関数

```
loadVariablesNum(url:String, level:Number [, method:String]) : Void
```

テキストファイルや、ColdFusion、CGI スクリプト、ASP、PHP、または Perl スクリプトで作成されたテキストなどの外部ファイルからデータを読み取り、Flash Player の特定のレベルの変数に値を設定します。この関数を使用して、現在の SWF ファイルの変数を新しい値で更新することもできます。

指定された URL にあるテキストは、標準の MIME 形式 *application/x-www-form-urlencoded* (CGI スクリプトで使用される標準形式) である必要があります。変数はいくつでも指定できます。たとえば、次の例では複数の変数が定義されています。

```
company=Macromedia&address=601+Townsend&city=San+Francisco&zip=94103
```

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、*url* は、呼び出し元の SWF ファイルと同じスープードメインに属している必要があります。スープードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、www.someDomain.com に存在する SWF ファイルは、store.someDomain.com に存在するソースからデータをロードできます。これは、どちらのファイルも同じスープードメイン someDomain.com に属しているからです。

Flash Player 7 以降で動作する SWF ファイルでは、*url* に、この呼び出しを発行する SWF ファイルと完全に同じドメインを使用する必要があります (『ActionScript ユーザーガイド』の「Flash Player のセキュリティ機能」を参照)。たとえば www.someDomain.com に置かれている SWF ファイルは、www.someDomain.com に置かれているソースからのみデータをロードできます。異なるドメインからデータをロードする場合は、SWF ファイルをホストするサーバーに "クロスドメインポリシーファイル" を置いておく必要があります。詳細については、『ActionScript ユーザーガイド』の「ドメイン間のデータロード許可について」を参照してください。

変数をターゲットムービークリップにロードするには、*loadVariablesNum()* の代わりに *loadVariables()* を使用します。

パラメータ

url:String - 変数が存在する絶対 URL または相対 URL。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、*url* は SWF ファイルと同じドメインに属している必要があります。詳細については、「説明」を参照してください。

level:Number - 変数を受け取る Flash Player のレベルを指定する整数。

method:String(オプション) - 変数を送信するための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送る変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないとときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、テキストファイル "params.txt" の情報を、Flash Player のレベル 2 にある SWF ファイルのメインタイムラインにロードします。テキストフィールドの変数名は、"params.txt" ファイルで宣言されている変数名と同じでなければなりません。*setInterval()* 関数は、SWF へのデータのロードの進行状況をチェックする場合に使用します。このスクリプトは、"params.txt" ファイル内の *done* という名前の変数をチェックします。

```
loadVariablesNum("params.txt", 2);
function checkParamsLoaded() {
  if (_level2.done == undefined) {
    trace("not yet.");
  } else {
    trace("finished loading. killing interval.");
    trace("-----");
  }
}
```

```
for (i in _level2) {  
    trace(i+": "+_level2[i]);  
}  
trace("-----");  
clearInterval(param_interval);  
}  
}  
}  
var param_interval = setInterval(checkParamsLoaded, 100);  
  
// Params.txt includes the following text  
var1="hello"&var2="goodbye"&done="done"
```

関連項目

[getURL 関数](#), [loadMovie 関数](#), [loadMovieNum 関数](#), [loadVariables 関数](#), [loadMovie \(MovieClip.loadMovie メソッド\)](#), [loadVariables \(MovieClip.loadVariables メソッド\)](#), [load \(LoadVars.load メソッド\)](#)

mbchr 関数

`mbchr(number)`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `String.fromCharCode()` メソッドを使用します。

ASCII コード番号をマルチバイト文字に変換します。

パラメータ

`number:Number` - マルチバイト文字に変換する数値。

関連項目

[fromCharCode \(String.fromCharCode メソッド\)](#)

mblength 関数

`mblength(string) : Number`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `String.length` プロパティを使用します。

マルチバイト文字のストリング長を返します。

パラメータ

`string:String` - 長さを調べるストリング。

戻り値

Number - マルチバイト文字のストリング長。

関連項目

[String.length \(String.length プロパティ\)](#)

mbord 関数

`mbord(character) : Number`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `String.charCodeAt()` メソッドを使用します。

指定された文字をマルチバイト文字コード番号に変換します。

パラメータ

`character:String - character` マルチバイト文字コード番号に変換する文字。

戻り値

Number - 変換された文字。

関連項目

[charCodeAt \(String.charCodeAt メソッド\)](#)

mbsubstring 関数

`mbsubstring(value, index, count) : String`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `String.substr()` メソッドを使用します。

マルチバイト文字ストリングから、新たにマルチバイト文字ストリングを抽出します。

パラメータ

`value:String - 抽出元のマルチバイト文字ストリング。`

`index:Number - 抽出する最初の文字の位置を表す数値。`

`count:Number - 抽出するストリングの文字数。インデックス文字は除きます。`

戻り値

String - マルチバイト文字ストリングから抽出されたストリング。

関連項目

[substr \(String.substr メソッド\)](#)

nextFrame 関数

`nextFrame() : Void`

次のフレームに再生ヘッドを送ります。

例

次の例では、ユーザーが右矢印キーまたは下矢印キーを押すと、再生ヘッドが次のフレームに進んで停止します。ユーザーが左矢印キーまたは上矢印キーを押すと、再生ヘッドは前のフレームに戻って停止します。リスナーは初期化されます。矢印キーが押されるまで待機し、`init` 変数を使用して、再生ヘッドがフレーム 1 に移動したときにリスナーが再定義されるのを防ぎます。

```
stop();

if (init == undefined) {
    someListener = new Object();
    someListener.onKeyDown = function() {
        if (Key.isDown(Key.LEFT) || Key.isDown(Key.UP)) {
            _level0.prevFrame();
        } else if (Key.isDown(Key.RIGHT) || Key.isDown(Key.DOWN)) {
            _level0.nextFrame();
        }
    };
    Key.addListener(someListener);
    init = 1;
}
```

関連項目

[prevFrame 関数](#)

nextScene 関数

`nextScene() : Void`

次のシーンのフレーム 1 に再生ヘッドを送ります。

例

次の例では、ユーザーが実行時に作成されたボタンをクリックすると、再生ヘッドが次のシーンのフレーム 1 に送られます。2 つのシーンを作成し、次の ActionScript をシーン 1 のフレーム 1 に入力します。

```

stop();

if (init == undefined) {
    this.createEmptyMovieClip("nextscene_mc", this.getNextHighestDepth());
    nextscene_mc.createTextField("nextscene_txt", this.getNextHighestDepth(), 200,
        0, 100, 22);
    nextscene_mc.nextscene_txt.autoSize = true;
    nextscene_mc.nextscene_txt.border = true;
    nextscene_mc.nextscene_txt.text = "Next Scene";
    this.createEmptyMovieClip("prevscene_mc", this.getNextHighestDepth());
    prevscene_mc.createTextField("prevscene_txt", this.getNextHighestDepth(), 00,
        0, 100, 22);
    prevscene_mc.prevscene_txt.autoSize = true;
    prevscene_mc.prevscene_txt.border = true;
    prevscene_mc.prevscene_txt.text = "Prev Scene";
    nextscene_mc.onRelease = function() {
        nextScene();
    };
}

prevscene_mc.onRelease = function() {
    prevScene();
};

init = true;
}

```

stop() アクションをシーン2のフレーム1に配置したことを確認してください。

関連項目

[prevScene 関数](#)

Number 関数

`Number(expression) : Number`

次に示すように、パラメータ *expression* を数値に変換し、値を返します。

- *expression*が数値の場合は、戻り値は *expression*です。
- *expression*がブール値の場合は、*expression*が `true` であれば戻り値は1となり、*expression*が `false` であれば戻り値は0となります。
- *expression*がストリングである場合、関数は、*expression*を `1.57505e-3` のような指数表現を伴う10進数として解析しようことがあります。
- *expression*が `Nan` の場合は、戻り値は `Nan` です。
- *expression*が `undefined` の場合の戻り値は次のようにになります。- Flash Player 6以前用にパブリッシュされたファイルでは、結果は0になります。- Flash Player 7以降用にパブリッシュされたファイルでは、結果は `Nan` になります。

パラメータ

expression:Object - 数値に変換される式。Ox で始まる数値やストリングは 16 進数として解釈されます。O で始まる数値やストリングは 8 進数として解釈されます。

戻り値

Number - 数値または NaN (非数)。

例

次の例では、実行時にステージでテキストフィールドが作成されます。

```
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
counter_txt.autoSize = true;
counter_txt.text = 0;
function incrementInterval():Void {
    var counter:Number = counter_txt.text;
    // Without the Number() function, Flash would concatenate the value instead
    // of adding values. You could also use "counter_txt.text++;" 
    counter_txt.text = Number(counter) + 1;
}
var intervalID:Number = setInterval(incrementInterval, 1000);
```

関連項目

[NaN 定数](#), [Number](#), [parseInt 関数](#), [parseFloat 関数](#)

Object 関数

Object([value]) : Object

空のオブジェクトを新規作成するか、指定された数値、ストリング、またはブール値をオブジェクトに変換します。このコマンドは、Object コンストラクタを使用してオブジェクトを作成することと同じです。「Object クラスのコンストラクタ」を参照してください。

パラメータ

value:Object (オプション) - 数値、ストリング、またはブール値。

戻り値

Object - オブジェクト。

例

次の例では、新しい空のオブジェクトが作成され、そのオブジェクトに値が格納されます。

```
var company:Object = new Object();
company.name = "Macromedia, Inc.";
company.address = "600 Townsend Street";
company.city = "San Francisco";
company.state = "CA";
company.postal = "94103";
for (var i in company) {
  trace("company."+i+" = "+company[i]);
}
```

関連項目

[Object](#)

on ハンドラ

```
on(mouseEvent:Object) { // your statements here }
```

アクションをトリガするマウスイベントまたはキー押下を指定します。

パラメータ

mouseEvent:Object - *mouseEvent* は、" イベント " と呼ばれるトリガです。イベントが発生すると、中括弧 ({}) 内でそれに続くステートメントが実行されます。次の値のすべてを、*mouseEvent* パラメータに指定できます。

- **press** - ポインタがボタン上にあるときにマウスボタンを押した場合。
メモ: このイベントは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。
- **release** - ポインタがボタン上にあるときにマウスボタンを離した場合。
- **releaseOutside** - ポインタがボタン上にあるときにマウスボタンを押し、そのままボタン領域の外側に移動してからボタンを離した場合。press イベントと dragOut イベントは両方とも、常に releaseOutside イベントより前に配置します。
メモ: このイベントは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。
- **rollOut** - ポインタがボタン領域の外側に移動した場合。
メモ: このイベントは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。
- **rollOver** - マウスポインタがボタン上に移動した場合。
- **dragOut** - ポインタがボタン上にあるときにマウスボタンを押し、そのままボタン領域の外側に移動した場合。
- **dragOver** - ポインタがボタン上にあるときにマウスボタンを押し、そのままボタンの外側に移動し、またボタン上に戻った場合。

- keyPress "<key>" - 指定されたキーを押した場合。パラメータの *key* の部分には、[アクション] パネルのコードヒントで示されているように、キー定数を指定します。このパラメータを使用してキー入力を取得できます。つまり、特定のキーのビルトインビヘイビアを上書きできます。ボタンは、アプリケーションの任意の場所(ステージ上またはステージ外)に配置できます。この技術には制約が1つあります。on() ハンドラを実行時に適用できないため、オーサリング時に適用する必要があります。必ず [制御]-[キーボードショートカットを無効] を選択してください。 [制御]-[ムービープレビュー] を使用してアプリケーションをテストすると、ビルトインビヘイビアを持つ特定のキーは上書きされません。

キー定数の一覧については、Key クラスを参照してください。

例

次のスクリプトでは、マウスを押したときに startDrag() 関数が実行され、マウスを離してオブジェクトをドロップしたときに on (release) 内のスクリプトが実行されます。

```
on (press) {
    startDrag(this);
}
on (release) {
    trace("X:"+this._x);
    trace("Y:"+this._y);
    stopDrag();
}
```

関連項目

[onClipEvent ハンドラ](#), [Key](#)

onClipEvent ハンドラ

```
onClipEvent(movieEvent:Object) { // your statements here }
```

ムービークリップの特定のインスタンスに対して定義されたアクションを起動します。

パラメータ

movieEvent:Object - *movieEvent* は、" イベント " と呼ばれるトリガです。イベントが発生すると、その後ろの中括弧({})内のステートメントが実行されます。次の値のすべてを、*movieEvent* パラメータに指定できます。

- load - ムービークリップがインスタンス化され、タイムラインに表示されると、アクションが即座に開始されます。
- unload - ムービークリップがタイムラインから削除された後、最初のフレームでアクションが開始します。Unload ムービークリップイベントに関連するアクションは、該当するフレームにアクションが割り当てられる前に処理されます。

- enterFrame - アクションはムービークリップのフレームレートで継続的にトリガれます。
enterFrame クリップイベントに関連するアクションは、該当するフレームに割り当てられているフレームアクションの前に処理されます。
- mouseMove - マウスを動かすたびに、アクションが開始されます。現在のマウスの位置を指定するには、_xmouse プロパティと _ymouse プロパティを使用します。
メモ: このイベントは、System.capabilities.hasMouse が true の場合のみ Flash Lite でサポートされます。
- mouseDown - 左マウスボタンを押すと、アクションが開始されます。
メモ: このイベントは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。
- mouseUp - 左マウスボタンを離すと、アクションが開始します。
メモ: このイベントは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。
- keyDown - キーを押すと、アクションが開始します。最後に押されたキーについての情報を取得するには、Key.getCode() を使用します。
- keyUp - キーを離すと、アクションが開始します。最後に押されたキーについての情報を取得するには、Key.getCode() メソッドを使用します。
- data - loadVariables() アクションまたは loadMovie() アクションによってデータを取得すると、アクションが開始されます。loadVariables() アクションで指定すると、data イベントは最後の変数がロードされたときの1回だけ発生します。loadMovie() アクションで指定すると、各データセクションが読み込まれるたびに、data イベントが繰り返し発生します。

例

次の例は、keyDown ムービーイベントで onClipEvent() を使用しており、ムービークリップまたはボタンにアタッチされるように設計されています。通常、keyDown ムービーイベントは Key オブジェクトのメソッドやプロパティと共に使用します。次のスクリプトでは、Key.getCode() を使用して、ユーザーが押したキーを調べます。押されたキーが Key.RIGHT プロパティと一致する場合は、再生ヘッドを次のフレームに送ります。押されたキーが Key.LEFT プロパティと一致する場合は、再生ヘッドを前のフレームに送ります。

```
onClipEvent (keyDown) {
  if (Key.getCode() == Key.RIGHT) {
    this._parent.nextFrame();
  } else if (Key.getCode() == Key.LEFT) {
    this._parent.prevFrame();
  }
}
```

次の例では、`load` および `mouseMove` ムービーイベントで `onClipEvent()` を使用します。
`_xmouse` プロパティと `_ymouse` プロパティは、マウスが移動するたびにマウスの位置を追跡します。これらの値は、実行時に作成されるテキストフィールドに表示されます。

```
onClipEvent (load) {  
    this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 100, 22);  
    coords_txt.autoSize = true;  
    coords_txt.selectable = false;  
}  
onClipEvent (mouseMove) {  
    coords_txt.text = "X:"+_root._xmouse+",Y:"+_root._ymouse;  
}
```

関連項目

[Key,_xmouse \(MovieClip._xmouse プロパティ\),_ymouse \(MovieClip._ymouse プロパティ\),on ハンドラ](#)

ord 関数

`ord(character) : Number`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String クラスのメソッドとプロパティを使用します。

文字を ASCII コード番号に変換します。

パラメータ

`character:String - ASCII コード番号に変換する文字。`

戻り値

`Number - 指定した文字の ASCII コード番号。`

関連項目

[String.charCodeAt \(String.charCodeAt メソッド\)](#)

parseFloat 関数

`parseFloat(string:String) : Number`

ストリングを浮動小数に変換します。この関数は、先頭から始まる数値の一部でない文字に達するまでストリング内の数値を読み取り、つまり解析して、結果を返します。ストリングが解析できる数値で始まっている場合、`parseFloat()` は `Nan` を返します。有効な整数の前の空白は、後続の非数值文字と同様に無視されます。

パラメータ

`string:String` - 読み込まれて浮動小数に変換されるストリング。

戻り値

`Number` - 数値または `NaN` (非数)。

例

次の例では、`parseFloat()` 関数を使用して各種数値を評価します。

```
trace(parseFloat("-2")); // output: -2
trace(parseFloat("2.5")); // output: 2.5
trace(parseFloat(" 2.5")); // output: 2.5
trace(parseFloat("3.5e6")); // output: 3500000
trace(parseFloat("foobar")); // output: NaN
trace(parseFloat("3.75math")); // output: 3.75
trace(parseFloat("0garbage")); // output: 0
```

関連項目

[NaN 定数](#), [parseInt 関数](#)

parseInt 関数

`parseInt(expression:String [, radix:Number]) : Number`

ストリングを整数に変換します。パラメータで指定されたストリングを数値に変換できない場合は `NaN` を返します。Ox から始まる整数は、16 進数と解釈されます。O から始まる整数、または基数として 8 を指定する整数は 8 進数と解釈されます。有効な整数の前の空白は、後続の非数値文字と同様に無視されます。

パラメータ

`expression:String` - 整数に変換されるストリング。

`radix:Number` (オプション) - 解析する数値の基数を表す整数。有効な値は、2 ~ 36 です。

戻り値

`Number` - 数値または `NaN` (非数)。

例

次の例では、`parseInt()` 関数を使用して各種の数値を評価します。

次の例では 3 を返します。

```
parseInt("3.5")
```

次の例では NaN を返します。

```
parseInt("bar")
```

次の例では 4 を返します。

```
parseInt("4foo")
```

次の例は 16 進数の変換例で、1016 を返します。

```
parseInt("0x3F8")
```

次の例は、オプションの *radix* パラメータを使用した 16 進数の変換例で、1000 を返します。

```
parseInt("3E8", 16)
```

次は 2 進数の変換例で、10(2 進数 1010 の 10 進表現)を返します。

```
parseInt("1010", 2)
```

次は 8 進数の解析例で、511(8 進数 777 の 10 進表現)を返します。

```
parseInt("0777")
```

```
parseInt("777", 8)
```

関連項目

[NaN 定数](#), [parseFloat 関数](#)

play 関数

`play() : Void`

タイムライン内で再生ヘッドを前へ進めます。

例

次の例では、`stop_mc` と `play_mc` というインスタンス名の 2 つのムービークリップインスタンスがステージにあります。ActionScript は、`stop_mc` ムービークリップインスタンスがクリックされると、SWF ファイルの再生を停止します。`play_mc` インスタンスがクリックされると、再生を再開します。

```
this.stop_mc.onRelease = function() {
    stop();
};

this.play_mc.onRelease = function() {
    play();
};

trace("frame 1");
```

関連項目

[gotoAndPlay 関数](#), [gotoAndPlay \(MovieClip.gotoAndPlay メソッド\)](#)

prevFrame 関数

`prevFrame() : Void`

前のフレームに再生ヘッドを送ります。現在のフレームが1である場合、再生ヘッドは移動しません。

例

次の ActionScript が myBtn_btn というボタンのタイムラインのフレームに配置されていると、ユーザーがこのボタンをクリックしたときに、再生ヘッドが前のフレームに送られます。

```
stop();
this.myBtn_btn.onRelease = function(){
    prevFrame();
};
```

関連項目

[nextFrame 関数](#), [prevFrame \(MovieClip.prevFrame メソッド\)](#)

prevScene 関数

`prevScene() : Void`

前のシーンのフレーム1に再生ヘッドを送ります。

関連項目

[nextScene 関数](#)

random 関数

`random(value) : Number`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `Math.random()` を使用します。

0 から `value` パラメータで指定された整数まで（この整数は含まない）の間のランダムな整数を返します。

パラメータ

`value:Number` - 整数。

戻り値

`Number` - ランダムな整数。

例

`random(5);` のように `random()` を使用すると、値 0、1、2、3、または 4 が返されます。

関連項目

[random \(Math.random メソッド\)](#)

removeMovieClip 関数

`removeMovieClip(target:Object)`

指定されたムービークリップを削除します。

パラメータ

`target:Object` - `duplicateMovieClip()` で作成したムービークリップインスタンスのターゲットパスか、`MovieClip.attachMovie()`、`MovieClip.duplicateMovieClip()`、または `MovieClip.createEmptyMovieClip()` で作成したムービークリップのインスタンス名。

例

次の例では、`myClip_mc` という新しいムービークリップを作成し、そのムービークリップを複製します。2番目のムービークリップは、`newClip_mc` となります。両方のムービークリップにイメージがロードされます。`button_mc` ボタンがクリックされると、複製されたムービークリップがステージから削除されます。

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
myClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(this.myClip_mc, "newClip_mc", this.getNextHighestDepth());
newClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
newClip_mc._x = 200;
this.button_mc.onRelease = function() {
    removeMovieClip(this._parent.newClip_mc);
};
```

関連項目

[duplicateMovieClip 関数](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#), [attachMovie \(MovieClip.attachMovie メソッド\)](#), [removeMovieClip \(MovieClip.removeMovieClip メソッド\)](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド\)](#)

setInterval 関数

```
setInterval(functionName:Object, interval:Number [, param1:Object, param2, ..., paramN]) : Number setInterval(objectName:Object, methodName:String, interval:Number [, param1:Object, param2, ..., paramN]) : Number
```

SWF ファイルの再生時に一定の間隔で関数、メソッド、またはオブジェクトを呼び出します。インターバル関数を使用して、データベースから変数を更新したり、時間表示を更新したりできます。

interval が SWF ファイルのフレームレートを超える場合は、画面の更新に伴う影響を最小限に抑えるために、インターバル関数は再生ヘッドがフレームに入ったときにのみ呼び出されます。

メモ : Flash Lite 2.0 では、インターバルが SWF ファイルのフレームレートより短い場合、このメソッドに渡されたインターバルは無視され、インターバル関数は、SWF ファイルのフレームレートの間隔でのみ呼び出されます。間隔が、SWF ファイルのフレームレートより長い場合、イベントは、間隔が経過した後の次のフレームで呼び出されます。

パラメータ

functionName:Object - 関数名または匿名関数への参照。

interval:Number - *functionName* または *methodName* パラメータに対する呼び出しの間隔(ミリ秒)。

param:Object (オプション) - *functionName* または *methodName* パラメータに渡すパラメータ。*param1, param2, ..., paramN* のように複数のパラメータはカンマで区切ります。

objectName:Object - *methodName* メソッドを含むオブジェクト。

methodName:String - *objectName* のメソッド。

戻り値

Number - 間隔を識別するための整数。これを *clearInterval()* に渡して間隔をクリアします。

例

シンタックス 1: 次の例では、1000 ミリ秒(1秒)ごとに匿名関数を呼び出します。

```
setInterval( function(){ trace("interval called"); }, 1000 );
```

シンタックス 2: 次の例では、2つのイベントハンドラを定義し、各イベントハンドラを呼び出します。最初の *setInterval()* の呼び出しが、*trace()* ステートメントを含む *callback1()* 関数を呼び出します。2番目の *setInterval()* の呼び出しが、ストリング "interval called" をパラメータとして *callback2()* 関数に渡します。

```
function callback1() {  
    trace("interval called");  
}  
function callback2() {  
    trace("interval called");  
}
```

```

function callback2(arg) {
  trace(arg);
}

setInterval( callback1, 1000 );
setInterval( callback2, 1000, "interval called" );

シンタックス 3:次の例では、オブジェクトのメソッドを使用します。オブジェクトに対して定義されたメソッドを呼び出す場合は、このシンタックスを使用する必要があります。

obj = new Object();
obj.interval = function() {
  trace("interval function called");
}

setInterval( obj, "interval", 1000 );

obj2 = new Object();
obj2.interval = function(s) {
  trace(s);
}
setInterval( obj2, "interval", 1000, "interval function called" );

オブジェクトのメソッドを呼び出すには、次の例に示すように、2つ目の形式の setInterval() シンタックスを使用する必要があります。

setInterval( obj2, "interval", 1000, "interval function called" );
この関数を使用する場合は、SWF ファイルで使用するメモリに注意する必要があります。たとえば、SWF ファイルからムービークリップを削除した場合、SWF ファイル内で実行される setInterval() 関数は削除されません。次の例に示すように、setInterval() 関数の使用が終了したたびに、clearInterval() を使用してこの関数を削除してください。

// create an event listener object for our MovieClipLoader instance
var listenerObject = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
  trace("start interval");
  /* after the target movie clip loaded, create a callback which executes
  about every 1000 ms (1 second) and calls the intervalFunc function. */
  target_mc.myInterval = setInterval(intervalFunc, 1000, target_mc);
};

function intervalFunc(target_mc) {
  // display a trivial message which displays the instance name and arbitrary
  // text.
  trace(target_mc+" has been loaded for "+getTimer()/1000+" seconds.");
  /* when the target movie clip is clicked (and released) you clear the interval
  and remove the movie clip. If you don't clear the interval before deleting
  the movie clip, the function still calls itself every second even though the
  movie clip instance is no longer present. */
  target_mc.onRelease = function() {
    trace("clear interval");
    clearInterval(this.myInterval);
  }
}

```

```
// delete the target movie clip
removeMovieClip(this);
};

}

var jpeg_mcl:MovieClipLoader = new MovieClipLoader();
jpeg_mcl.addListener(listenerObject);
jpeg_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
this.createEmptyMovieClip("jpeg_mc", this.getNextHighestDepth()));

setInterval() をクラス内で使用する場合は、この関数を呼び出すときに this キーワードを使用していることを確認する必要があります。このキーワードを使用しないと、setInterval() 関数がクラスメンバーにアクセスできません。次に、このことを説明する例を示します。deleteUser_btn, というボタンを含む FLA ファイルで、次の ActionScript をフレーム 1 に追加します。
```

```
var me:User = new User("Gary");
this.deleteUser_btn.onRelease = function() {
trace("Goodbye, "+me.username);
 clearInterval(me.intervalID);
 delete me;
};
```

次に、FLA ファイルと同じディレクトリに "User.as" というファイルを作成します。次の ActionScript を入力します。

```
class User {
var intervalID:Number;
var username:String;
function User(param_username:String) {
trace("Welcome, "+param_username);
this.username = param_username;
this.intervalID = setInterval(this, "traceUsername", 1000, this.username);
}
function traceUsername(str:String) {
trace(this.username+" is "+getTimer()/1000+" seconds old, happy birthday.");
}
}
```

関連項目

[clearInterval 関数](#)

setProperty 関数

`setProperty(target:Object, property:Object, expression:Object) : Void`

ムービークリップの再生時にムービークリップのプロパティ値を変更します。

パラメータ

`target:Object` - 設定対象のプロパティがあるムービークリップのインスタンス名へのパス。

`property:Object` - 設定するプロパティ。

`expression:Object` - プロパティの新しいリテラル値またはプロパティの新しい値として評価される等式のいずれか一方。

例

次の ActionScript は、新しいムービークリップを作成し、イメージをロードします。setProperty() を使用して、そのクリップの _x 座標と _y 座標を設定します。right_btn というボタンをクリックすると、params_mc というムービークリップの _x 座標が 20 ピクセル増加します。

```
this.createEmptyMovieClip("params_mc", 999);
params_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
setProperty(this.params_mc, _y, 20);
setProperty(this.params_mc, _x, 20);
this.right_btn.onRelease = function() {
  setProperty(params_mc, _x, getProperty(params_mc, _x)+20);
};
```

関連項目

[getProperty 関数](#)

startDrag 関数

`startDrag(target:Object [, lock:Boolean, left:Number, top:Number, right:Number, bottom:Number]) : Void`

ムービーの再生中に target ムービークリップをドラッグ可能にします。一度に1つのムービークリップしかドラッグできません。startDrag() を実行した後は、stopDrag() によって明示的に停止するか、他のムービークリップに対して startDrag() アクションが呼び出されるまで、ムービークリップはドラッグ可能なままになります。

メモ: このメソッドは、`System.capabilities.hasMouse` が `true` である、または `System.capabilities.hasStylus` が `true` の場合のみ Flash Lite でサポートされます。

パラメータ

`target:Object` - ドラッグするムービークリップのターゲットパス。

lock:Boolean (オプション) - ドラッグ可能なムービークリップがマウス位置の中心にロックされるか (true)、ユーザーがムービークリップ上で最初にクリックした点にロックされるか (false) を指定するブール値。

left,top,right,bottom:Number (オプション) - ムービークリップの制限矩形を指定するムービークリップの親の座標を基準にした相対値。

例

次の例では、ムービークリップ pic_mc を作成し、そのムービークリップに startDrag() アクションと stopDrag() アクションを追加することで、実行時にユーザーが任意の位置にドラッグできるようになります。イメージは、MovieClipLoader クラスを使用して pic_mc にロードされます。

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
  target_mc.onPress = function() {
    startDrag(this);
  };
  target_mc.onRelease = function() {
    stopDrag();
  };
};
pic_mcl.addListener(listenerObject);
```

関連項目

[stopDrag 関数](#), [_droptarget \(MovieClip._droptarget プロパティ\)](#), [startDrag \(MovieClip.startDrag メソッド\)](#)

stop 関数

stop() : *Void*

再生中の SWF ファイルを停止します。このアクションは、ボタンでムービークリップを制御する場合に最もよく使用します。

関連項目

[gotoAndStop 関数](#), [gotoAndStop \(MovieClip.gotoAndStop メソッド\)](#)

stopAllSounds 関数

`stopAllSounds() : Void`

再生ヘッドを停止せずに、SWF ファイルで再生中のサウンドをすべて停止します。ストリーミングするために設定されたサウンドは、そのサウンドが置かれているフレームに再生ヘッドが移動すると再生を再開します。

例

次のコードを実行するとテキストフィールドが作成され、そこに音楽の ID3 情報が表示されます。新しい Sound オブジェクトインスタンスが作成され、SWF ファイルに MP3 がロードされます。サウンドファイルから ID3 情報が抽出されます。ユーザーが stop_mc をクリックすると、サウンドが一時停止します。ユーザーが play_mc をクリックすると、一時停止されていた位置から再生が再開します。

```
this.createTextField("songinfo_txt", this.getNextHighestDepth, 0, 0,
    Stage.width, 22);
var bg_sound:Sound = new Sound();
bg_sound.loadSound("yourSong.mp3", true);
bg_sound.onID3 = function() {
    songinfo_txt.text = "(" + this.id3.artist + " ) " + this.id3.album + " - " +
        this.id3.track + " - "
    + this.id3.songname;
    for (prop in this.id3) {
        trace(prop+ " = "+this.id3[prop]);
    }
    trace("ID3 loaded.");
};
this.play_mc.onRelease = function() {
/* get the current offset. if you stop all sounds and click the play button, the
   MP3 continues from
   where it was stopped, instead of restarting from the beginning. */
    var numSecondsOffset:Number = (bg_sound.position/1000);
    bg_sound.start(numSecondsOffset);
};
this.stop_mc.onRelease = function() {
    stopAllSounds();
};
```

関連項目

[Sound](#)

stopDrag 関数

`stopDrag() : Void`

現在実行中のドラッグ操作を停止します。

メモ: このメソッドは、`System.capabilities.hasMouse` が `true` である、または `System.capabilities.hasStylus` が `true` の場合のみ Flash Lite でサポートされます。

例

次のコードをメインタイムラインに置くと、ユーザーがマウスボタンを離したときにムービークリップインスタンス `my_mc` でのドラッグアクションが停止します。

```
my_mc.onPress = function () {
    startDrag(this);
}

my_mc.onRelease = function() {
    stopDrag();
}
```

関連項目

[startDrag 関数](#), [_droptarget \(MovieClip._droptarget プロパティ\)](#), [startDrag \(MovieClip.startDrag メソッド\)](#), [stopDrag \(MovieClip.stopDrag メソッド\)](#)

String 関数

`String(expression:Object) : String`

指定されたパラメータのストリング表現を返します。具体的には次の処理が行われます。

- `expression` が数値である場合、返されるストリングは数値のテキスト表現です。
- `expression` がストリングの場合は、返されるストリングは `expression` です。
- `expression` がオブジェクトである場合、戻り値は、オブジェクトのストリングプロパティを呼び出した結果として生成されるストリング表現です。このようなプロパティが存在しない場合は、`Object.toString()` を呼び出した結果として生成されるストリング表現です。
- `expression` がブール値である場合、返されるストリングは `"true"` または `"false"` です。
- `expression` がムービークリップである場合、戻り値はスラッシュ (/) 表記のムービークリップのターゲットパスです。

`expression` が `undefined` である場合は、戻り値は次のようにになります。

- Flash Player 6 以前用にパブリッシュされたファイルでは、結果は空のストリング ("") になります。
- Flash Player 7 以降用にパブリッシュされたファイルでは、結果は `undefined` になります。

メモ: スラッシュ表記は、ActionScript 2.0 ではサポートされていません。

パラメータ

expression:Object - ストリングに変換される式。

戻り値

String - ストリング。

例

次の例では ActionScript を使用して、指定された式をストリングに変換します。

```
var string1:String = String("3");
var string2:String = String("9");
trace(string1+string2); // output: 39
```

どちらのパラメータもストリングの場合、値は加算されるのではなく連結されます。

関連項目

[toString \(Number.toString メソッド\)](#), [toString \(Object.toString メソッド\)](#),
[String, " ストリング区切り記号演算子](#)

substring 関数

substring("string", index, count) : String

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.substr() を使用します。

ストリングの一部を抽出します。この関数は 1 から始まります。一方、String オブジェクトのメソッドは 0 から始まります。

パラメータ

string:String - 新しいストリングを抽出する元のストリング。

index:Number - 抽出する最初の文字の位置を表す数値。

count:Number - 抽出するストリングの文字数。インデックス文字は除きます。

戻り値

String - 抽出されたサブストリング。

関連項目

[substr \(String.substr メソッド\)](#)

targetPath 関数

`targetpath(targetObject:Object) : String`

MovieClip、Button、またはTextField オブジェクトのターゲットパスをストリングとして返します。ターゲットパスは、ドット(.) 表記で返されます。スラッシュ(/) 表記でターゲットパスを検索するには、_target プロパティを使用します。

パラメータ

`targetObject:Object` - 検索するターゲットパスのオブジェクトへの参照(_root や _parent など)。MovieClip、Button、またはTextField オブジェクトなどを指定できます。

戻り値

`String` - 指定したオブジェクトのターゲットパスを含むストリング。

例

次の例では、ムービークリップをロードすると同時にそのターゲットパスを表示します。

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
trace(targetPath(myClip_mc)); // _level0.myClip_mc
```

関連項目

[eval 関数](#)

tellTarget 関数

`tellTarget("target") { statement(s); }`

非推奨 Flash Player 5 以降では使用しないでください。代わりにドット(.) 表記と with ステートメントを使用することをお勧めします。

`statements` パラメータで指定された指示を、`target` パラメータで指定されたタイムラインに適用します。tellTarget アクションは、ナビゲーションコントロールに使用すると便利です。ステージ上のムービークリップを停止または開始するボタンに、tellTarget を割り当てます。ムービークリップを、そのクリップ内の特定のフレームにジャンプさせることもできます。たとえば、ボタンに tellTarget を指定して、ステージ上のムービークリップを停止または開始したり、特定のフレームにジャンプしたりするように指示することができます。

Flash 5 以降では、tellTarget アクションの代わりにドット(.) 表記を使用できます。with アクションを使用して同じタイムラインに対して複数のアクションを発行できます。with アクションのターゲットは任意のオブジェクトですが、tellTarget アクションのターゲットはムービークリップのみです。

パラメータ

target:String - 制御するタイムラインのターゲットパスを指定するストリング。

statement(s):Object - 条件が true である場合に実行される指示。

例

次の tellTarget ステートメントでは、メインタイムライン上のムービークリップインスタンス ball を制御します。ball インスタンスのフレーム 1 は空白であり、stop() アクションがあるので、ステージ上では見えません。次のアクションを備えたボタンがクリックされると、tellTarget は ball 内の再生ヘッドに対して、アニメーションが始まるフレーム 2 に進むように指示します。

```
on(release) {  
    tellTarget("_parent.ball") {  
        gotoAndPlay(2);  
    }  
}
```

次の例では、ドット(.) 表記を使用して同じことを行います。

```
on(release) {  
    _parent.ball.gotoAndPlay(2);  
}
```

ball インスタンスに対して複数のコマンドを発行する場合は、次のステートメントに示すように with アクションを使用できます。

```
on(release) {  
    with(_parent.ball) {  
        gotoAndPlay(2);  
        _alpha = 15;  
        _xscale = 50;  
        _yscale = 50;  
    }  
}
```

関連項目

[with ステートメント](#)

toggleHighQuality 関数

`toggleHighQuality()`

非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに `_quality` を使用します。

Flash Player でアンチエイリアス処理のオンとオフを切り替えます。アンチエイリアス処理を行うとオブジェクトのエッジが滑らかになりますが、ムービーの再生は遅くなります。これは、Flash Player のすべての SWF ファイルに影響します。

例

次のコードは、アンチエイリアス処理のオンとオフをクリックで切り替えるボタンに適用できます。

```
on(release) {  
    toggleHighQuality();  
}
```

関連項目

[_highquality プロパティ](#), [_quality プロパティ](#)

trace 関数

`trace(expression:Object)`

Flash Debug Player を使用すると、`trace()` 関数の出力を取得し、その結果をログファイルに書き込むことができます。

ステートメント。式を評価し、その結果をプレビューモードで [出力] パネルに表示します。

このステートメントは、ムービーのプレビュー中にプログラミングのメモを記録したり [出力] パネルにメッセージを表示する場合に使用します。条件の有無を確認したり、値を [出力] パネルに表示したりするには、`expression` パラメータを使用します。`trace()` ステートメントは、JavaScript の `alert` 関数に似ています。

[パブリッシュ設定] ダイアログボックスで [Trace アクションを省略] コマンドを使用すると、書き出す SWF ファイルから `trace()` アクションを削除することができます。

パラメータ

`expression:Object` - 評価する式。[ムービープレビュー] コマンドを使用して SWF ファイルを Flash オーサリングツールで開くと、`expression` パラメータの値が [出力] パネルに表示されます。

例

次の例では、trace()ステートメントを使用して、動的に作成されるテキストフィールドerror_txtのメソッドとプロパティを[出力]パネルに表示します。

```
this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
for (var i in error_txt) {
    trace("error_txt."+i+" = "+error_txt[i]);
}
/* output:
error_txt.styleSheet = undefined
error_txt.mouseWheelEnabled = true
error_txt.condenseWhite = false
...
error_txt.maxscroll = 1
error_txt.scroll = 1
*/
```

unescape 関数

`unescape(x:String) : String`

パラメータ *x* をストリングとして評価し、URL エンコードされた形式からストリングをデコード（すべての16進シーケンスを ASCII 文字に変換）して、ストリングを返します。

パラメータ

`string:String` - アンエスケープする16進シーケンスのストリング。

戻り値

`String` - URL エンコードされたパラメータからデコードしたストリング。

例

次の例では、エスケープからアンエスケープへの変換処理を示します。

```
var email:String = "user@somedomain.com";
trace(email);
var escapedEmail:String = escape(email);
trace(escapedEmail);
var unescapedEmail:String = unescape(escapedEmail);
trace(unescapedEmail);
```

次の結果が[出力]パネルに表示されます。

```
user@somedomain.com
user%40somedomain%2Ecom
user@somedomain.com
```

unloadMovie 関数

`unloadMovie(target:MovieClip) : Void` `unloadMovie(target:String) : Void`

`loadMovie()` を使ってロードしたムービークリップを Flash Player から削除します。

`loadMovieNum()` を使ってロードしたムービーをアンロードするには、`unloadMovie()` ではなく `unloadMovieNum()` を使用します。

パラメータ

`target` - ムービークリップのターゲットパス。このパラメータには、`String ("my_mc" など)`、またはムービークリップインスタンス (`my_mc など`)への直接参照のいずれかを指定できます。複数のデータ型を指定できるパラメータは、`Object` としてリストされます。

例

次の例では、`pic_mc` という名前で新しいムービークリップを作成し、そのクリップにイメージをロードします。イメージは `MovieClipLoader` クラスを使用してロードします。イメージをクリックすると、ムービークリップが SWF ファイルからアンロードされます。

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
  target_mc.onRelease = function() {
    unloadMovie(pic_mc);
    /* or you could use the following, which refers to the movie clip referenced by
     'target_mc'. */
    //unloadMovie(this);
  };
};
pic_mcl.addListener(listenerObject);
```

関連項目

`loadMovie (MovieClip.loadMovie メソッド)`,
`unloadClip (MovieClipLoader.unloadClip メソッド)`

unloadMovieNum 関数

`unloadMovieNum(level:Number) : Void`

`loadMovieNum()` を使用してロードした SWF ファイルやイメージを Flash Player から削除します。`MovieClip.loadMovie()` を使用してロードした SWF またはイメージをアンロードするには、`unloadMovieNum()` ではなく `unloadMovie()` を使用します。

パラメータ

`level:Number` - ロードされたムービーのレベル (`_level N`)。

例

次の例では、SWF ファイルにイメージをロードします。`unload_btn` をクリックすると、ロードされたコンテンツが削除されます。

```
loadMovieNum("yourimage.jpg", 1);
unload_btn.onRelease = function() {
    unloadMovieNum(1);
}
```

関連項目

[loadMovieNum 関数](#), [unloadMovie 関数](#), [loadMovie \(MovieClip.loadMovie メソッド\)](#)

グローバルプロパティ

グローバルプロパティは、どのスクリプトでも使用でき、ドキュメント内のどのタイムラインやスコープでも参照できます。たとえば、グローバルプロパティを使用して、ロードされた他のムービークリップのタイムラインに相対アクセス (`_parent`) または絶対アクセス (`_root`) できます。また、スコープを制限 (`this`) または拡張 (`super`) することもできます。さらに、グローバルプロパティを使用して、スクリーンリーダーのアクセシビリティ、再生品質、サウンドバッファサイズなどの実行時設定を調整ができます。

グローバルプロパティ一覧

オプション	プロパティ	説明
	<code>\$version</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.version</code> プロパティを使用します。 Flash Lite のバージョン番号が格納されます。
	<code>_cap4WayKeyAS</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.has4WayKeyAS</code> プロパティを使用します。 右、左、上、下矢印キーに関連付けられたキーイベントハンドラに割り当てられた ActionScript の式を、Flash Lite が実行できるかどうかを示します。

オプション	プロパティ	説明
	<code>_capCompoundSound</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasCompoundSound</code> プロパティを使用します。 Flash Lite でコンパウンドサウンドデータを処理できるかを示します。
	<code>_capEmail</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasEmail</code> プロパティを使用します。 Flash Lite クライアントが <code>GetURL()</code> ActionScript コマンドを使用して、電子メールメッセージを送信できるかどうかを示します。
	<code>_capLoadData</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasDataLoading</code> プロパティを使用します。 ホストアプリケーションが <code>loadMovie()</code> 、 <code>loadMovieNum()</code> 、 <code>loadVariables()</code> 、および <code>loadVariablesNum()</code> 関数の呼び出しによって、動的に追加のデータをロードできるかどうかを示します。
	<code>_capMFi</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasMFi</code> プロパティを使用します。 デバイスが MFi (Melody Format for i-mode) オーディオ形式でサウンドデータを再生できるかどうかを示します。
	<code>_capMIDI</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasMIDI</code> プロパティを使用します。 デバイスが MIDI (Musical Instrument Digital Interface) オーディオ形式でサウンドデータを再生できるかどうかを示します。

オプション	プロパティ	説明
	<code>_capMMS</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasMMS</code> プロパティを使用します。 Flash Lite が MMS (Multimedia Messaging Service) メッセージを <code>GetURL()</code> ActionScript コマンドを使用して送信できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。
	<code>_capSMAF</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasSMAF</code> プロパティを使用します。 デバイスがマルチメディアファイルを SMAF (Synthetic music Mobile Application Format) で再生できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。
	<code>_capSMS</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasSMS</code> プロパティを使用します。 Flash Lite が <code>GetURL()</code> ActionScript コマンドを使用して SMS (Short Message Service) メッセージを送信できるかどうかを示します。
	<code>_capStreamSound</code>	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに <code>System.capabilities.hasStreamingAudio</code> プロパティを使用します。 デバイスがストリーミング (同期) サウンドを再生できるかどうかを示します。
	<code>_focusrect</code>	プロパティ (グローバル)。キーボードフォーカスがあるボタンまたはムービークリップの周囲に黄色の矩形を表示するかどうかを指定します。
	<code>_forceframerate</code>	指定したフレームレートでレンダリングするように Flash Lite プレーヤーに指示します。
	<code>_global</code>	<code>String</code> 、 <code>Object</code> 、 <code>Math</code> 、 <code>Array</code> などのコア ActionScript クラスを保持するグローバルオブジェクトへの参照です。

オプション	プロパティ	説明
	<code>_highquality</code>	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに <code>_quality</code> を使用します。現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。
	<code>_level</code>	<code>_level N</code> のルートタイムラインへの参照です。
	<code>maxscroll</code>	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに <code>TextField.maxscroll</code> を使用します。テキストフィールド内の最終行も表示されている場合に、そのフィールドの先頭に表示されるテキストの行番号を示します。
	<code>_parent</code>	現在のムービークリップまたはオブジェクトを含むムービークリップまたはオブジェクトへの参照を指定するか返します。
	<code>_quality</code>	ムービークリップに使用するレンダリング品質を設定または取得します。
	<code>_root</code>	ルートムービークリップのタイムラインへの参照を指定するか返します。
	<code>scroll</code>	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに <code>TextField.scroll</code> を使用します。変数に関連するテキストフィールドの情報表示を制御します。
	<code>_soundbuftime</code>	ストリーミングサウンドをバッファする秒数を設定します。
	<code>this</code>	オブジェクトインスタンスまたはムービークリップインスタンスを参照します。

\$version プロパティ

`$version`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.version` プロパティを使用します。

ストリング変数。Flash Lite のバージョン番号が格納されます。この変数には、メジャー番号、マイナー番号、ビルド番号、および内部ビルド番号が格納されます。リリースバージョンの場合、内部ビルド番号は通常は 0 です。すべての Flash Lite 1.x 製品のメジャー番号は 5 です。Flash Lite 1.0 はマイナー番号 1 を持ち、Flash Lite 1.1 はマイナー番号 2 を持ちます。

例

Flash Lite 1.1 プレーヤーでは、次のコードによって `myVersion` の値が "5, 2, 12, 0" に設定されます。

```
myVersion = $version;
```

関連項目

[version \(capabilities.version プロパティ\)](#)

_cap4WayKeyAS プロパティ

`_cap4WayKeyAS`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.has4WayKeyAS` プロパティを使用します。

数値変数。右、左、上、下矢印キーに関連付けられたキーイベントハンドラに割り当てられた ActionScript の式を、Flash Lite が実行できるかどうかを示します。ホストアプリケーションが 4 方向のキーナビゲーションモードを使用して Flash コントロール(ボタンおよび入力テキストフィールド)間を移動する場合のみ、この変数が定義され、値が 1 に設定されます。それ以外の場合、この変数は定義されません。

`_cap4WayKeyAS` の値が 1 の場合、4 方向のキーのいずれかが押されると、Flash Lite は最初にそのキーのハンドラを探します。ハンドラが見つからない場合、Flash コントロールナビゲーションが発生します。しかし、イベントハンドラが見つかった場合は、そのキーのナビゲーションアクションは発生しません。たとえば、下方向キーの `keypress` ハンドラが見つかった場合、ユーザーは移動できません。

例

次の例では、`canUse4Way` が Flash Lite 1.1 では 1 に設定されていますが、Flash Lite 1.0 では未定義です(ただし、Flash Lite 1.1 を使用したすべての携帯端末が 4 方向キーをサポートするわけではないため、このコードは携帯端末に依存します)。

```
canUse4Way = _cap4WayKeyAS;
if (canUse4Way == 1) {
    msg = "Use your directional joypad to navigate this application";
} else {
    msg = "Please use the 2 key to scroll up, the 6 key to scroll right,
the 8 key to scroll down, and the 4 key to scroll left.";
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capCompoundSound プロパティ

`_capCompoundSound`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに

`System.capabilities.hasCompoundSound` プロパティを使用します。

数値変数。Flash Lite がコンパウンドサウンドデータを処理できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。たとえば、1 つの Flash ファイルに同じサウンドの MIDI と MFi の両方の形式を含めることができます。Player は、デバイスがサポートする形式に基づいて適切な形式でデータを再生します。この変数は、Flash Lite プレーヤーが現在の携帯端末でこの機能サポートするかどうかを定義します。

例

次の例では、`useCompoundSound` が Flash Lite 1.1 では 1 に設定されていますが、Flash Lite 1.0 では未定義です。

```
useCompoundSound = _capCompoundSound;

if (useCompoundSound == 1) {
    gotoAndPlay("withSound");
} else {
    gotoAndPlay("withoutSound");
```

関連項目

[capabilities \(System.capabilities\)](#)

_capEmail プロパティ

`_capEmail`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに

`System.capabilities.hasEmail` プロパティを使用します。

数値変数。Flash Lite クライアントが `GetURL()` ActionScript コマンドを使用して電子メールメッセージを送信できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、ホストアプリケーションが `GetURL()` ActionScript コマンドを使用して電子メールメッセージを送信できる場合、`canEmail()` を 1 に設定します。

```
canEmail = _capEmail;

if (canEmail == 1) {
```

```
getURL("mailto:someone@somewhere.com?subject=foo&body=bar");
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capLoadData プロパティ

`_capLoadData`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasDataLoading` プロパティを使用します。

数値変数。ホストアプリケーションが `loadMovie()`、`loadMovieNum()`、`loadVariables()`、および `loadVariablesNum()` 関数の呼び出しを通じて動的に追加データをロードできるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、ホストアプリケーションがムービーと変数を動的にロードできる場合、`CanLoad` を 1 に設定します。

```
canLoad = _capLoadData;

if (canLoad == 1) {
    loadVariables("http://www.somewhere.com/myVars.php", GET);
} else {
    trace ("client does not support loading dynamic data");
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capMFi プロパティ

`_capMFi`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasMFi` プロパティを使用します。

数値変数。デバイスが MFi (Melody Format for i-mode) オーディオ形式でサウンドデータを再生できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、デバイスが MFi サウンドデータを再生できる場合、canMFi を 1 に設定します。

```
canMFi = _capMFi;

if (canMFi == 1) {
    // send movieclip buttons to frame with buttons that trigger events

    sounds
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

関連項目

[hasMFI \(capabilities.hasMFI プロパティ\)](#)

_capMIDI プロパティ

[_capMIDI](#)

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasMIDI` プロパティを使用します。

数値変数。デバイスが MIDI (Musical Instrument Digital Interface) オーディオ形式でサウンドデータを再生できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、デバイスが MIDI サウンドデータを再生できる場合、_capMIDI を 1 に設定します。

```
canMIDI = _capMIDI;

if (canMIDI == 1) {
    // send movieclip buttons to frame with buttons that trigger events

    sounds
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capMMS プロパティ

_capMMS

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasMMS` プロパティを使用します。

数値変数。Flash Lite が MMS (Multimedia Messaging Service) メッセージを `GetURL()` ActionScript コマンドを使用して送信できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、Flash Lite 1.1 の場合は `canMMS` は 1 に設定されますが、Flash Lite 1.0 の場合は未定義のままになります（ただし、すべての Flash Lite 1.1 の携帯端末が MMS メッセージを送信できるわけではないので、このコードは携帯端末に依存します）。

```
on(release) {  
    canMMS = _capMMS;  
    if (canMMS == 1) {  
        // send an MMS  
        myMessage = "mms:4156095555?body=sample mms message";  
    } else {  
        // send an SMS  
        myMessage = "sms:4156095555?body=sample sms message";  
    }  
    getURL(myMessage);  
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capSMAF プロパティ

_capSMAF

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasSMAF` プロパティを使用します。

数値変数。デバイスが SMAF (Synthetic music Mobile Application Format) 形式のマルチメディアファイルを再生できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、Flash Lite 1.1 の場合は canSMAF は 1 に設定されますが、Flash Lite 1.0 の場合は未定義のままになります（ただし、すべての Flash Lite 1.1 の携帯端末が SMAF メッセージを送信できるわけではないので、このコードは携帯端末に依存します）。

```
canSMAF = _capSMAF;

if (canSMAF) {
    // send movieclip buttons to frame with buttons that trigger events

    sounds
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capSMS プロパティ

`_capSMS`

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに `System.capabilities.hasSMS` プロパティを使用します。

数値変数。Flash Lite が `GetURL()` ActionScript コマンドを使用して SMS (Short Message Service) メッセージを送信できるかどうかを示します。できる場合は、この変数が定義され、値が 1 に設定されます。できない場合、この変数は定義されません。

例

次の例では、`canSMS` は Flash Lite 1.1 では 1 に設定されますが、Flash Lite 1.0 では未定義のままになります（ただし、すべての Flash Lite 1.1 の携帯端末が SMS メッセージを送信できるわけではないので、このコードは携帯端末に依存します）。

```
on(release) {
    canSMS = _capSMS;
    if (canSMS) {
        // send an SMS
        myMessage = "sms:4156095555?body=sample sms message";
        getURL(myMessage);
    }
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_capStreamSound プロパティ

_capStreamSound

非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに

`System.capabilities.hasStreamingAudio` プロパティを使用します。

数値変数。デバイスがストリーミング(同期)サウンドを再生できるかどうかを示します。できる場合は、この変数が定義され、値が1に設定されます。できない場合、この変数は定義されません。

例

次の例では、`canStreamSound` が有効な場合にストリーミングサウンドを再生します。

```
on(press) {  
    canStreamSound = _capStreamSound;  
    if (canStreamSound) {  
        // play a streaming sound in a movieclip with this button  
        tellTarget("music") {  
            gotoAndPlay(2);  
        }  
    }  
}
```

関連項目

[capabilities \(System.capabilities\)](#)

_focusrect プロパティ

_focusrect = Boolean;

キーボードフォーカスがあるボタンまたはムービークリップの周囲に黄色の矩形を表示するかどうかを指定します。`_focusrect` がデフォルト値の `true` に設定されている場合、ユーザーが `Tab` キーを押して SWF ファイル内のオブジェクト間を移動するときに、現在フォーカスのあるボタンまたはムービークリップの周囲に黄色の矩形が表示されます。黄色の矩形を表示しない場合は、`false` を指定します。これはプロパティであり、特定のインスタンスに対して上書きできます。

グローバル `_focusrect` プロパティが `false` に設定されている場合、すべてのボタンとムービークリップのデフォルトのビヘイビアでは、キーボードナビゲーションが `Tab` キーに限定されます。`Enter` キーや矢印キーを含め、他のキーはすべて無視されます。すべてのキーナビゲーションを元に戻すには、`_focusrect` を `true` に設定する必要があります。特定のボタンやムービークリップのキー操作機能をすべて回復させる場合は、`Button._focusrect` と `MovieClip._focusrect` のいずれを使用しても、このグローバルプロパティを上書きできます。

メモ: コンポーネントを使用する場合、`FocusManager` は、このグローバルプロパティの使用を含め、Flash Player のフォーカス処理を上書きします。

メモ : Flash Lite 2.0 プレーヤーでは、_focusrect プロパティが無効になっている場合(つまり、`Button.focusRect = false` や `MovieClip.focusRect = false` などの場合)、ボタンまたはムービークリップはすべてのイベントを受け取ります。この動作は Flash Player とは異なります。Flash Player では _focusrect プロパティが無効になっている場合、ボタンまたはムービークリップは、`rollOver` イベントおよび `rollOut` イベントを受け取ますが、`press` イベントおよび `release` イベントは受け取りません。

また、Flash Lite 2.0 では、`fscommand2 SetFocusRectColor` コマンドを使用して、フォーカス矩形のカラーを変更できます。この動作は Flash Player とは異なります。Flash Player では、フォーカス矩形の色は黄色に制限されます。

例

次の例では、ブラウザウィンドウでフォーカスを受け取ったときに、SWF ファイル内のインスタンスの周囲の黄色の矩形を非表示にします。ボタンまたはムービークリップをいくつか作成し、次の ActionScript をタイムラインのフレーム 1 に追加します。

```
_focusrect = false;
```

関連項目

[_focusrect \(Button._focusrect プロパティ\)](#), [_focusrect \(MovieClip._focusrect プロパティ\)](#)

_forceframeRate プロパティ

`_forceframeRate`

`true` に設定されている場合、このプロパティは、指定したフレームレートでレンダリングするように Flash Lite プレーヤーに指示します。コンテンツにデバイスサウンドが含まれている場合、このプロパティを擬似同期サウンドで使用できます。このプロパティは、デフォルトで `false` に設定されており、Flash Lite は通常どおりにレンダリングします。`true` に設定されている場合、Flash Lite プレーヤーは、フレームレートを維持するために特定のフレームをスキップすることができます。

_global プロパティ

`_global.identifier`

`String`、`Object`、`Math`、`Array` などのコア ActionScript クラスを保持するグローバルオブジェクトへの参照です。たとえば、`Math` オブジェクトや `Date` オブジェクトのように、グローバル ActionScript オブジェクトとして公開されるライブラリを作成できます。タイムライン宣言またはローカル宣言した変数や関数とは異なり、グローバル変数および関数は SWF ファイルのすべてのタイムラインおよびスコープに公開されます。ただし、ネストされているスコープで同じ名前の識別子により隠蔽される場合を除きます。

メモ : グローバル変数の値を設定するとき、_global.variableName など、変数の完全修飾名を使用する必要があります。そうしないと、設定するグローバル変数を不明瞭なものとする、同じ名前のローカル変数が作成されます。

戻り値 String、Object、Math、Array などのコア ActionScript クラスを保持するグローバルオブジェクトへの参照です。

例

次の例では、SWF ファイルのすべてのタイムラインとスコープで使用できるトップレベル関数 factorial(), を作成します。

```
_global.factorial = function(n:Number) {
    if (n<=1) {
        return 1;
    } else {
        return n*factorial(n-1);
    }
}
// Note: factorial 4 == 4*3*2*1 == 24
trace(factorial(4)); // output: 24
```

次の例では、グローバル変数の値を設定するときに完全修飾変数名を誤って使用すると、予期しない結果がどのように発生するかを示します。

```
_global.myVar = "global";
trace("_global.myVar: " + _global.myVar); // _global.myVar: global
trace("myVar: " + myVar); // myVar: global

myVar = "local";
trace("_global.myVar: " + _global.myVar); // _global.myVar: global
trace("myVar: " + myVar); // myVar: local
```

関連項目

[var ステートメント](#) , [set variable ステートメント](#)

_highquality プロパティ

`_highquality`

非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに `_quality` を使用します。

現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。最高品質を適用するには、2(最高品質)を指定します。アンチエイリアス処理を適用するには、1(高品質)を指定します。アンチエイリアス処理を避けるには、0(低品質)を指定します。

例

次の ActionScript をメインタイムラインに指定し、アンチエイリアス処理が適用されるようにグローバルな quality プロパティを設定します。`_highquality = 1;`

関連項目

[_quality プロパティ](#)

_level プロパティ

`_levelN`

`_levelN` のルートタイムラインへの参照です。`_level` プロパティを使用してターゲットとする SWF ファイルは、`loadMovieNum()` を使用して Flash Player 内に事前にロードしておく必要があります。`_level N` を使用すると、ロードされている SWF ファイルを `N` によって割り当てられたレベルのターゲットにすることもできます。

Flash Player のインスタンス内にロードした最初の SWF ファイルは、自動的に `_level10` 内にロードされます。`_level10` の SWF ファイルによって、その後にロードするすべての SWF ファイルのフレームレート、背景色、およびフレームサイズが設定されます。次に、SWF ファイルは `_level10` の SWF ファイルより高い番号のレベルに積み重ねられます。

Flash Player 内にロードする SWF ファイルごとに、`loadMovieNum()` アクションを使用してレベルを割り当てる必要があります。レベルは任意の順番で割り当てることができます。SWF ファイルが既に含まれているレベル (`_level10` を含む) を割り当てるとき、そのレベルの SWF ファイルはアンロードされ、新しい SWF ファイルに置き換えられます。

例

次の例では、`_level9` にロードされた SWF ファイル "sub.swf" のメインタイムラインで再生ヘッドを停止します。"sub.swf" ファイルにはアニメーションが含まれています。このファイルは、次の ActionScript を含むドキュメントと同じディレクトリにあります。

```
loadMovieNum("sub.swf", 9);
myBtn_btn.onRelease = function() {
    _level9.stop();
};
```

上記の例の `_level9.stop()` は、次のコードで置き換えることができます。

```
_level9.gotoAndStop(5);
```

このアクションは、再生ヘッドを停止する代わりに、`_level9` の SWF ファイルのメインタイムラインからフレーム 5 に再生ヘッドを送ります。

関連項目

[loadMovie 関数](#), [swapDepths \(MovieClip.swapDepths メソッド\)](#)

maxscroll プロパティ

variable_name.maxscroll

非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに

TextField.maxscroll を使用します。

テキストフィールド内の最終行も表示されている場合に、そのフィールドの先頭に表示されるテキストの行番号を表示します。maxscroll プロパティは、scroll プロパティと連携してテキストフィールド内の情報表示を制御します。このプロパティは読み取り専用であり、変更できません。

関連項目

[maxscroll \(TextField.maxscroll プロパティ\)](#), [scroll \(TextField.scroll プロパティ\)](#)

_parent プロパティ

_parent.property

_parent._parent.property

現在のムービークリップまたはオブジェクトを含むムービークリップまたはオブジェクトへの参照を指定するか返します。現在のオブジェクトとは、_parent を参照する ActionScript コードがあるオブジェクトです。現在のムービークリップまたはオブジェクトの上位のムービークリップまたはオブジェクトへの相対パスを指定するには、_parent を使用します。

例

次の例では、square_mc というインスタンス名のムービークリップがステージにあります。このムービークリップ内には、circle_mc というインスタンス名の別のムービークリップがあります。次の ActionScript では、円をクリックすると circle_mc 親インスタンス (square_mc) を変更できます。相対アドレスを使用する場合 (_root の代わりに _parent を使用) は、最初に [アクション] パネルの [ターゲットパスを挿入] ボタンを使用する方が簡単です。

```
this.square_mc.circle_mc.onRelease = function() {
    this._parent._alpha -= 5;
};
```

関連項目

[_root プロパティ](#), [targetPath 関数](#)

_quality プロパティ

_quality:String

ムービークリップに使用するレンダリング品質を設定または取得します。デバイスフォントは常にエイリアス処理されるため、_quality プロパティには影響されません。

_quality プロパティは、次の値に設定できます。

値	説明	グラフィックのアンチエイリアス	ビットマップスミング
"LOW"	低いレンダリング品質。この設定は、テキストを含まないムービーに適しています。	グラフィックスはアンチエイリアス処理されていません。	ビットマップはスムージングされません。
"MEDIUM"	普通のレンダリング品質。この設定は、テキストを含まないムービーに適しています。	グラフィックスは 2×2 ピクセルグリッドを使用してアンチエイリアス処理されます。	ビットマップはスムージングされません。
"HIGH"	高いレンダリング品質。デフォルトのレンダリング品質設定です。	グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理されます。	ビットマップはスムージングされません。

例

次の例では、レンダリング品質を LOW に設定します。

```
_quality = "LOW";
```

_root プロパティ

*_root.movieClip
_root.action
_root.property*

ルートムービークリップのタイムラインへの参照を指定するか返します。ムービークリップに複数のレベルがある場合、ルートムービークリップのタイムラインは現在実行中のスクリプトを含むレベルにあります。たとえば、レベル1のスクリプトの評価が _root であれば、_level1 が返されます。

_root を指定することは、使用を避けるべきスラッシュ表記 (/) を使って現在のレベル内の絶対パスを指定することと同じです。

メモ : _root を含むムービークリップを他のムービークリップにロードすると、_root は _root を含むムービークリップのタイムラインではなく、ロードする側のムービークリップのタイムラインを指すようになります。他のムービークリップにロードする場合でも、_root がロードされるムービークリップ自身のタイムラインを指すようにするには、MovieClip._lockroot を使用します。

パラメータ

movieClip:String - ムービークリップのインスタンス名。
action:String - アクションまたはフィールド。
property:String - MovieClip オブジェクトのプロパティ。

例

次の例では、現在実行中のスクリプトを含むレベルのタイムラインを停止します。

```
_root.stop();
```

次の例では、_root のスコープ内の変数とインスタンスを出力します。

```
for (prop in _root) {  
    trace("_root."+prop+" = "+_root[prop]);  
}
```

関連項目

[_lockroot \(MovieClip._lockroot プロパティ\)](#), [_parent プロパティ](#), [targetPath 関数](#)

scroll プロパティ

textFieldVariableName.scroll = x

非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに `TextField.scroll` を使用します。

変数に関連するテキストフィールドの情報表示を制御します。`scroll` プロパティは、テキストフィールドにおける内容の表示開始位置を定義します。設定後、Flash Player はユーザーがテキストフィールドをスクロールするたびに、内容を更新します。`scroll` プロパティは、長い文節内の特定の段落にユーザーを誘導したり、スクロールテキストフィールドを作成したりする場合に便利です。このプロパティは、取得および変更が可能です。

例

次の例は、テキストフィールド `myText` をスクロールする UP ボタンに割り当てるコードです。

```
on (release) {  
    myText.scroll = myText.scroll + 1;  
}
```

関連項目

[maxscroll \(TextField.maxscroll プロパティ\)](#), [scroll \(TextField.scroll プロパティ\)](#)

_soundbuftime プロパティ

`_soundbuftime:Number = integer`

ストリーミングサウンドをバッファする秒数を設定します。デフォルト値は 5 秒です。

パラメータ

`integer:Number` - SWF ファイルのストリーミングが開始するまでの秒数。

例

次の例では、MP3 ファイルをストリーミングし、サウンドをバッファした後でユーザーに対して再生します。実行時にテキストフィールドが 2 つ作成され、タイマー情報とデバッグ情報が収められます。`_soundbuftime` プロパティは、MP3 を 10 秒間バッファするように設定されます。MP3 用に新しい Sound オブジェクトインスタンスが作成されます。

```
// create text fields to hold debug information.  
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100, 22);  
this.createTextField("debug_txt", this.getNextHighestDepth(), 0, 20, 100, 22);  
// set the sound buffer to 10 seconds.  
_soundbuftime = 10;  
// create the new sound object instance.  
var bg_sound:Sound = new Sound();  
// load the MP3 sound file and set streaming to true.  
bg_sound.loadSound("yourSound.mp3", true);  
// function is triggered when the song finishes loading.  
bg_sound.onLoad = function() {  
    debug_txt.text = "sound loaded";  
};  
debug_txt.text = "sound init";  
function updateCounter() {  
    counter_txt.text++;  
}  
counter_txt.text = 0;  
setInterval(updateCounter, 1000);
```

this プロパティ

`this`

オブジェクトインスタンスまたはムービークリップインスタンスを参照します。スクリプトの実行時には、`this` はそのスクリプトを含むムービークリップインスタンスを参照します。フィールドの実行時には、`this` には呼び出されたフィールドを含むオブジェクトへの参照が入ります。

ボタンに割り当てられた `on()` イベントハンドラ内では、`this` はそのボタンを含むタイムラインを参照します。ムービークリップに割り当てられている `onClipEvent()` イベントハンドラ内では、`this` はそのムービークリップ自体のタイムラインを参照します。

`this` は、それが指定されたスクリプトのコンテキスト内で評価されます。したがって、クラスファイルで定義された変数については、スクリプト内で `this` を指定して参照することはできません。`"ApplyThis.as"` という名前のファイルを作成し、次のコードを入力します。

```
class ApplyThis {  
    var str:String = "Defined in ApplyThis.as";  
    function concatStr(x:String):String {  
        return x+x;  
    }  
    function addStr():String {  
        return str;  
    }  
}
```

次に、FLA ファイルまたは AS ファイル内に、次の ActionScript を入力します。

```
var obj:ApplyThis = new ApplyThis();  
var abj:ApplyThis = new ApplyThis();  
abj.str = "defined in FLA or AS";  
trace(obj.addStr.call(abj, null)); //output: defined in FLA or AS  
trace(obj.addStr.call(this, null)); //output: undefined  
trace(obj.addStr.call(obj, null)); //output: Defined in applyThis.as
```

同様に、ダイナミッククラスで定義された関数を呼び出すには、`this` を使用して適切なスコープで関数を呼び出す必要があります。

```
// incorrect version of Simple.as  
/*  
dynamic class Simple {  
    function callfunc() {  
        trace(func());  
    }  
}  
*/  
// correct version of Simple.as  
dynamic class simple {  
    function callfunc() {  
        trace(this.func());  
    }  
}
```

FLA ファイルまたは AS ファイル内に、次の ActionScript を入力します。

```
var obj:Simple = new Simple();  
obj.num = 0;  
obj.func = function() {  
    return true;  
};  
obj.callfunc();  
// output: true
```

不適切なバージョンの `"Simple.as"` を使用するとシンタックスエラーが発生します。

例

次の例では、キーワード `this` は `Circle` オブジェクトを参照します。

```
function Circle(radius:Number):Void {  
    this.radius = radius;  
    this.area = Math.PI*Math.pow(radius, 2);  
}  
var myCircle = new Circle(4);  
trace(myCircle.area);
```

ムービークリップ内のフレームに割り当てられた次のステートメントでは、キーワード `this` は現在のムービークリップを参照します。

```
// sets the alpha property of the current movie clip to 20  
this._alpha = 20;
```

`MovieClip.onPress` ハンドラ内の次のステートメントでは、キーワード `this` は現在のムービークリップを参照します。

```
this.square_mc.onPress = function() {  
    startDrag(this);  
};  
this.square_mc.onRelease = function() {  
    stopDrag();  
};
```

関連項目

[on ハンドラ](#), [onClipEvent ハンドラ](#)

演算子

記号演算子は式の値の組み合わせ、比較、または修正の方法を指定する文字です。

演算子一覧

演算子	説明
<code>+</code> (<code>addition</code>)	数値式を加算するか、ストリングを連結(結合)します。
<code>+=</code> (<code>addition assignment</code>)	<code>expression1</code> に <code>expression1 + expression2</code> の値を代入します。
<code>[]</code> (<code>array access</code>)	指定されたエレメント(<code>a0</code> など)で新しい配列または多次元配列を初期化するか、配列内のエレメントにアクセスします。
<code>=</code> (<code>assignment</code>)	<code>expression2</code> の値(右側のパラメータ)を <code>expression1</code> の変数、配列エレメント、またはプロパティに代入します。
<code>&</code> (<code>bitwise AND</code>)	<code>expression1</code> と <code>expression2</code> を32ビット符号なし整数に変換し、整数パラメータをビット単位で論理積(AND)演算します。

演算子	説明
<code>&= (bitwise AND assignment)</code>	<code>expression1</code> に <code>expression1 & expression2</code> の値を代入します。
<code><< (bitwise left shift)</code>	<code>expression1</code> と <code>expression2</code> を 32 ビット整数に変換し、 <code>expression2</code> の変換により生成された整数で指定される桁数だけ <code>expression1</code> 内のすべてのビットを左にシフトします。
<code><<= (bitwise left shift and assignment)</code>	この演算子はビット単位の左シフト (<code><<</code>) 演算を行い、その内容を結果として <code>expression1</code> に格納します。
<code>~ (bitwise NOT)</code>	1 の補数演算子またはビット単位の補数演算子とも呼ばれます。
<code> (bitwise OR)</code>	<code>expression1</code> と <code>expression2</code> を 32 ビット符号なし整数に変換し、 <code>expression1</code> と <code>expression2</code> の対応ビットの少なくとも一方が 1 である各ビット位置に 1 を返します。
<code> = (bitwise OR assignment)</code>	<code>expression1</code> に <code>expression1 expression2</code> の値を代入します。
<code>>> (bitwise right shift)</code>	<code>expression1</code> と <code>expression2</code> を 32 ビット整数に変換し、 <code>expression2</code> の変換により生成された整数で指定される桁数だけ <code>expression1</code> 内のすべてのビットを右にシフトします。
<code>>>= (bitwise right shift and assignment)</code>	この演算子はビット単位の右シフト演算を行い、その内容を結果として <code>expression1</code> に格納します。
<code>>>> (bitwise unsigned right shift)</code>	左側のビットには常に 0 が詰められるために元の <code>expression</code> の符号が 保持されないという点を除いて、ビット単位の右シフト (<code>>></code>) 演算子と同じ です。浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。
<code>>>>= (bitwise unsigned right shift and assignment)</code>	ビット単位の符号なし右シフト演算を行い、その内容を結果として <code>expression1</code> に格納します。
<code>^ (bitwise XOR)</code>	<code>expression1</code> と <code>expression2</code> を 32 ビット符号なし整数に変換し、 <code>expression1</code> と <code>expression2</code> の対応ビットのいずれか一方のみが 1 である各ビット位置に 1 を返します。
<code>^= (bitwise XOR assignment)</code>	<code>expression1</code> に <code>expression1 ^ expression2</code> の値を代入します。
<code>/* (block comment delimiter)</code>	スクリプトコメントのブロックを示します。
<code>, (comma)</code>	式 <code>expression1</code> 、式 <code>expression2</code> 、... の順に評価します。

演算子	説明
add (concatenation (strings))	<p>非推奨 Flash Player 5 以降では使用しないでください。Flash Player 5 以降用のコンテンツを作成する場合は、加算 (+) 演算子を使用することをお勧めします。</p> <p>メモ : Flash Lite 2.0 では、add 演算子も使用されなくなりました。代わりに加算 (+) 演算子を使用します。</p> <p>複数のストリングを連結します。</p>
?: (conditional)	<i>expression1</i> を評価し、 <i>expression1</i> の値が true である場合は <i>expression2</i> の値を返します。それ以外の場合は <i>expression3</i> の値を返します。
-- (decrement)	<i>expression</i> から1を引くプリデクリメント単項演算子またはポストデクリメント単項演算子です。
/ (division)	<i>expression1</i> を <i>expression2</i> で除算します。
/= (division assignment)	<i>expression1</i> に <i>expression1 / expression2</i> の値を代入します。
. (dot)	ネストされた子のムービークリップ、変数、またはプロパティにアクセスするためにムービークリップの階層をナビゲートする場合に使用します。
== (equality)	2つの式の等価性をテストします。
eq (equality (strings))	<p>非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに == (等価) 演算子を使用します。</p> <p><i>expression1</i>のストリング表現が <i>expression2</i>のストリング表現と等しい場合は true を返します。それ以外の場合は false を返します。</p>
> (greater than)	2つの式を比較し、 <i>expression1</i> が <i>expression2</i> より大きいかどうかを判定します。大きい場合は true を返します。
gt (greater than (strings))	<p>非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに > (より大きい) 演算子を使用します。</p> <p><i>expression1</i>のストリング表現を <i>expression2</i>のストリング表現と比較し、<i>expression1</i>が <i>expression2</i>より大きい場合は true を返します。それ以外の場合は false を返します。</p>
>= (greater than or equal to)	2つの式を比較し、 <i>expression1</i> が <i>expression2</i> より大きいか等しいか(true)、または <i>expression1</i> が <i>expression2</i> より小さいか(false)を判定します。
ge (greater than or equal to (strings))	<p>非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに >= (より大きい、または等しい) 演算子を使用します。</p> <p><i>expression1</i>が <i>expression2</i>より大きいか等しい場合は true を返します。それ以外の場合は false を返します。</p>
++ (increment)	<i>expression</i> に1を加えるプリインクリメント単項演算子またはポストインクリメント単項演算子です。

演算子	説明
<code>!= (inequality)</code>	等価(==)演算子の正反対が真であるかどうかをテストします。
<code><> (inequality)</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子は使用されなくなりました。 <code>!= (inequality)</code> 演算子を使用することをお勧めします。 等価(==)演算子の正反対が真であるかどうかをテストします。
<code>instanceof</code>	<code>object</code> が <code>classConstructor</code> のインスタンスまたは <code>classConstructor</code> のサブクラスであるかどうかをテストします。
<code>< (less than)</code>	2つの式を比較し、 <code>expression1</code> が <code>expression2</code> より小さいかどうかを判定します。小さい場合は <code>true</code> を返します。
<code>lt (less than (strings))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code><</code> (より小さい) 演算子を使用します。 <code>expression1</code> が <code>expression2</code> よりも小さい場合は <code>true</code> を返します。それ以外の場合は <code>false</code> を返します。
<code><= (less than or equal to)</code>	2つの式を比較し、 <code>expression1</code> が <code>expression2</code> よりも小さいか等しいかどうかを判定します。小さいか等しい場合は <code>true</code> を返します。
<code>le (less than or equal to (strings))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code><=</code> (より小さい、または等しい) 演算子を使用します。 <code>expression1</code> が <code>expression2</code> より小さいか等しい場合は <code>true</code> を返します。それ以外の場合は <code>false</code> を返します。
<code>// (line comment delimiter)</code>	スクリプトコメントの先頭を示します。
<code>&& (logical AND)</code>	オペランドをブール(論理)値として評価し、論理演算を行います。
<code>and (logical AND)</code>	非推奨 Flash Player 5 以降では使用しないでください。論理積(&&)演算子を使用することをお勧めします。 Flash Player 4 で論理積(AND)(&&)演算を行います。
<code>! (logical NOT)</code>	変数や式のブール値を反転します。
<code>not (logical NOT)</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>! (logical NOT)</code> 演算子を使用します。 Flash Player 4 で論理否定(NOT)(!)演算を行います。
<code> (logical OR)</code>	オペランドをブール(論理)値として評価し、論理演算を行います。オペランドがブール値を返す論理式(等価や非等価、比較などを調べる式)であれば、両式が共に <code>false</code> の場合のみ <code>false</code> を返します。それ以外の場合、つまりどちらか一方の式または両式が <code>true</code> であれば、戻り値は <code>true</code> になります。

演算子	説明
or (logical OR)	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに (論理和) 演算子を使用します。 <i>condition1</i> と <i>condition2</i> を評価し、いずれかの式が true であれば、式全体が true になります。
% (modulo)	<i>expression1</i> を <i>expression2</i> で割ったときの剰余を計算します。
%= (modulo assignment)	<i>expression1</i> に <i>expression1 % expression2</i> の値を代入します。
* (multiplication)	2つの数値や式を乗算します。
*= (multiplication assignment)	<i>expression1</i> に <i>expression1 * expression2</i> の値を代入します。
new	新しい匿名のオブジェクトを作成し、constructor パラメータで指定された関数を呼び出します。
ne (not equal (strings))	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに != (inequality) 演算子を使用します。 <i>expression1</i> が <i>expression2</i> と等しくない場合は true を返します。 それ以外の場合は false を返します。
{ } (object initializer)	新しいオブジェクトを作成し、指定された name と value プロパティペアで初期化します。
() (parentheses)	パラメータに対してグループ化演算を実行するか、複数の式を順番に評価します。または、パラメータを囲み、結果をパラメータとして括弧の外側にある関数に渡します。
===(strict equality)	2つの式が等しいかどうかをテストします。厳密な等価 (==) 演算子は、データ型が変換されない点を除いては、等価 (==) 演算子と同じです。
!= (strict inequality)	厳密な等価 (==) 演算子の正反対が真であるかどうかをテストします。
" (string delimiter)	引用符 ("") で前後を囲んだ文字はリテラル値を表します。変数や数値、その他の ActionScript エレメントではなく、ストリングと見なされます。
- (subtraction)	符号反転や減算に使用します。
-= (subtraction assignment)	<i>expression1</i> に <i>expression1 - expression2</i> の値を代入します。
: (type)	厳密な型指定を行う際に使用します。この演算子では、変数のタイプ、関数の戻り値のタイプ、または関数パラメータのタイプを指定します。
typeof	typeof 演算子は <i>expression</i> を評価し、その式が String、MovieClip、Object、Function、Number、Boolean のいずれの値であるかを示すストリングを返します。
void	void 演算子は式を評価した後、その値を破棄して、undefined を返します。

+ 加算演算子

expression1 + expression2

数値式を加算するか、ストリングを連結(結合)します。ある式がストリングの場合、他の式はすべてストリングに変換され、連結されます。両方の式が整数の場合、合計は整数になります。いずれかの式または両方の式が浮動小数の場合、合計は浮動小数になります。

メモ: Flash Lite 2.0 では、加算(+)演算子を使用して、数値式を加算し、ストリングを連結することができます。Flash Lite 1.x では、加算(+)演算子を使用して、数値式の加算(" var1 = 1 + 2 // output: 3"など)のみを行うことができます。Flash Lite 1.x では、add 演算子を使用してストリングを連結する必要があります。

オペランド

expression1 - 数値またはストリング。

expression2 - 数値またはストリング。

戻り値

Object - ストリング、整数、または浮動小数。

例

シンタックス1:次の例では、2つのストリングを連結し、その結果を[出力]パネルに表示します。

```
var name:String = "Cola";
var instrument:String = "Drums";
trace(name + " plays " + instrument); // output: Cola plays Drums
```

メモ: Flash Lite 1.x では、加算(+)演算子を使用してストリングを連結することはできません。Flash Lite 1.x では、add 演算子を使用してストリングを連結する必要があります。

シンタックス2:このステートメントでは、整数2と3を加算し、結果の整数5を[出力]パネルに表示します。

```
trace(2 + 3); // output: 5
```

このステートメントでは、浮動小数2.5と3.25を加算し、結果の浮動小数5.75を[出力]パネルに表示します。

```
trace(2.5 + 3.25); // output: 5.75
```

シンタックス3:動的フィールドおよびテキスト入力フィールドに関連付けられた変数のデータ型はストリングです。次の例では、変数depositはステージのテキスト入力フィールドです。ユーザーが預金額を入力すると、スクリプトはdepositをoldBalanceに加算しようとします。ただし、depositはStringデータ型であるため、スクリプトは変数の値を合計せずに値を連結(結合して1つのストリングを構成)します。

```
var oldBalance:Number = 1345.23;
var currentBalance = deposit_txt.text + oldBalance;
trace(currentBalance);
```

たとえば、ユーザーが預金テキストフィールドに「475」と入力すると、trace() 関数は値 4751345.23 を [出力] パネルに表示します。これを修正するには、次のように Number() 関数を使用してストリングを数値に変換します。

```
var oldBalance:Number = 1345.23;
var currentBalance:Number = Number(deposit_txt.text) + oldBalance;
trace(currentBalance);
```

次の例から、文字列式の右側にある数値の合計が計算されないことがわかります。

```
var a:String = 3 + 10 + "asdf";
trace(a); // 13asdf
var b:String = "asdf" + 3 + 10;
trace(b); // asdf310
```

+= 加算後代入演算子

expression1 += *expression2*

expression1 に *expression1+ expression2* の値を代入します。たとえば、次の 2 つのステートメントは同じ結果になります。

```
x += y;
x = x + y;
```

この演算子は、ストリングの連結も行います。加算 (+) 演算子のすべての規則が、加算して代入 (+=) 演算子に適用されます。

オペランド

expression1 : Number - 数値またはストリング。

expression2 : Number - 数値またはストリング。

戻り値

Number - 加算した結果。

例

シンタックス 1: 次の例では、+= 演算子をストリング式で使用しています。"My name is Gilbert" が [出力] パネルに表示されます。

```
var x1:String = "My name is ";
x1 += "Gilbert";
trace(x1); // output: My name is Gilbert
```

シンタックス 2: 次に、数値に対して加算後代入 (+=) 演算子を使用する例を示します。

```
var x:Number = 5;
var y:Number = 10;
x += y;
trace(x); // output: 15
```

関連項目

+ [加算演算子](#)

[] 配列アクセス演算子

```
myArray = [ a0, a1,...aN ]
myArray[ i ] = value
myObject [ propertyName ]
```

指定されたエレメント (*a0* など) で新しい配列または多次元配列を初期化するか、配列内のエレメントにアクセスします。配列アクセス演算子を使用すると、インスタンス、変数、およびオブジェクト名を動的に設定および取得できます。この演算子を使用してオブジェクトのプロパティにアクセスすることもできます。

シンタックス1: 配列は、エレメントと呼ばれるプロパティを持つオブジェクトです。各エレメントは、インデックスと呼ばれる番号で識別されます。配列を作成する場合は、エレメントを配列アクセス演算子 ([])(角括弧)で囲みます。配列には、さまざまなタイプのエレメントを格納できます。たとえば、次の配列 *employee* には 3 つのエレメントがあります。最初のエレメントは数値、2 番目と 3 番目のエレメントはストリング(引用符内)です。

```
var employee:Array = [15, "Barbara", "Jay"];
```

括弧をネストすると、多次元配列をシミュレートできます。ネスト可能な配列の深さは、256 レベルまでです。次のコードでは、3 つのエレメントで構成される配列 *ticTacToe* を作成します。各エレメントはそれぞれが 3 つのエレメントで構成される配列になっています。

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; // Select Debug > List
    Variables in test mode
// to see a list of the array elements.
```

シンタックス2: 各エレメントに直接アクセスするには、エレメントのインデックスを [](角括弧)で囲みます。また、配列に新しいエレメントを追加したり、既存のエレメントの値を変更または取得することもできます。次の例に示すように、配列の最初のインデックスは常に 0 です。

```
var my_array:Array = new Array();
my_array[0] = 15;
my_array[1] = "Hello";
my_array[2] = true;
```

次の例のように角括弧 ([]) を使用して 4 番目のエレメントを追加できます。

```
my_array[3] = "George";
```

角括弧 ([]) を使用して多次元配列のエレメントにアクセスできます。最初の [](角括弧) のセットは元の配列のエレメントを示します。2 番目のセットはネストされた配列のエレメントを示します。次のコード行は、数値 6 を [出力] パネルに表示します。

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
trace(ticTacToe[1][2]); // output: 6
```

シンタックス 3: eval() 関数の代わりに配列アクセス演算子 ([]) を使用して、ムービークリップ名の値またはオブジェクトのプロパティを動的に設定および取得できます。次のコード行は、数値 6 を [出力] パネルに表示します。

```
name["mc" + i] = "left_corner";
```

オペランド

myArray : Object - myArray 配列の名前。

a0, a1,...aN : Object - a0,a1,...aN 配列のエレメント。ネストされた配列を含む、すべてのネイティブタイプまたはオブジェクトインスタンスです。

i : Number - i 0 以上の整数インデックス。

myObject : Object - myObject オブジェクトの名前。

propertyName : String - propertyName オブジェクトのプロパティを指定するストリング。

戻り値

Object -

シンタックス 1: 配列への参照。

シンタックス 2: 配列に格納されたいずれかの値。ネイティブタイプまたはオブジェクトインスタンス (Array インスタンスを含む)。

シンタックス 3: オブジェクトのいずれかのプロパティ。ネイティブタイプまたはオブジェクトインスタンス (Array インスタンスを含む)。

例

次の例では、新しい空の Array オブジェクトを作成する 2 つの方法を示しています。最初の行で角括弧 ([]) を使用しています。

```
var my_array:Array = [];
var my_array:Array = new Array();
```

次の例では、配列 employee_array を作成し、trace() ステートメントを使用して、エレメントを [出力] パネルに表示します。4 行目で配列内のエレメントを変更し、5 行目では変更後の新しい配列を [出力] パネルに表示しています。

```
var employee_array = ["Barbara", "George", "Mary"];
trace(employee_array); // output: Barbara,George,Mary
employee_array[2] = "Sam";
trace(employee_array); // output: Barbara,George,Sam
```

次の例では、[](角括弧)内の式("piece" + i)を評価し、その結果をムービークリップ my_mc から取得する変数の名前として使用します。この例では、変数 i はボタンと同じタイムライン上にある必要があります。たとえば、変数 i の値が 5 である場合は、ムービークリップ my_mc の変数 piece5 の値が [出力] パネルに表示されます。

```
myBtn_btn.onRelease = function() {  
    x = my_mc["piece"+i];  
    trace(x);  
};
```

次のコードでは、[](角括弧)内の式を評価し、その結果をムービークリップ name_mc から取得する変数の名前として使用します。

```
name_mc["A" + i];
```

Flash 4 の ActionScript スラッシュシンタックスの使用法に詳しい場合は、`eval()` 関数を使用して同じ結果を得ることができます。

```
eval("name_mc.A" & i);
```

次の ActionScript を使用すると、_root スコープ内のすべてのオブジェクトを繰り返し処理できます。デバッグ時に効果的な方法です。

```
for (i in _root) {  
    trace(i+": "+_root[i]);  
}
```

代入ステートメントの左側で配列アクセス ([]) 演算子を使用して、インスタンス、変数、およびオブジェクト名を動的に設定することもできます。

```
employee_array[2] = "Sam";
```

関連項目

[Array](#), [Object](#), [eval](#) 関数

= 代入演算子

expression1 = expression2

expression2 の値(右側のパラメータ)を *expression1* の変数、配列エレメント、またはプロパティに代入します。値による代入と、参照による代入があります。値による代入では、*expression2* の実際の値がコピーされ、*expression1* に格納されます。値による代入を使用した場合、変数には数値またはストリングのリテラルが代入されます。参照による代入では、*expression2* への参照が *expression1* に格納されます。一般に、参照による代入は、`new` 演算子と組み合わせて使用されます。`new` 演算子を使用した場合、メモリ内にオブジェクトが作成され、そのオブジェクトが格納されたメモリ位置への参照が変数に代入されます。

オペランド

expression1 : Object - 変数、配列のエレメント、またはオブジェクトのプロパティ。

expression2 : Object - 任意のタイプの値。

戻り値

Object - 代入された値、*expression2*。

例

次の例では、値による代入を使用して、値 5 を変数 x に代入しています。

```
var x:Number = 5;
```

次の例では、値による代入を使用して、値 "hello" を変数 x に代入しています。

```
var x:String;x = " hello ";
```

次の例では、参照による代入を使用して、moonsOfJupiter 変数を作成しています。この変数には、新たに作成された Array オブジェクトへの参照が格納されます。次に値による代入を使用し、"Callisto" という値を、変数 moonsOfJupiter で参照される配列の最初のエレメントにコピーしています。

```
var moonsOfJupiter:Array = new Array();moonsOfJupiter[0] = "Callisto";
```

次の例では、参照による代入を使用して新しいオブジェクトを作成し、変数 mercury にこのオブジェクトへの参照を格納しています。次に、値による代入を使用して、mercury オブジェクトの diameter プロパティに値 3030 を代入しています。

```
var mercury:Object = new Object(); mercury.diameter = 3030; // in miles  
trace (mercury.diameter); // output: 3030
```

次の例は、上記の例の続きです。merkur (mercury のドイツ語) という変数を作成し、その変数に mercury の値を代入しています。これにより、メモリ内の同じオブジェクトを参照する 2 つの変数が作成されました。同じオブジェクトを参照しているので、どちらの変数を使用しても、このオブジェクトのプロパティにアクセスできます。次に、マイルの代わりにキロメートルを使用するように、diameter プロパティを変更することもできます。

```
var merkur:Object = mercury; merkur.diameter = 4878; // in kilometers  
trace (mercury.diameter); // output: 4878
```

関連項目

[== 等価演算子](#)

& ビット単位の論理積 (AND) 演算子

expression1 & expression2

expression1 と *expression2* を 32 ビット符号なし整数に変換し、整数パラメータをビット単位で論理積 (AND) 演算します。浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。結果は、新しい 32 ビット整数です。

正の整数は 4294967295 (0xFFFFFFFF) を最大値とする符号なし 16 進値に変換されます。最大値より大きい値は、32 ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2 の補数表現を使用して、-2147483648 (0x800000000) を最小値とする符号なし 16 進値に変換されます。最小値よりも小さい値は、さらに高い精度で 2 の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの 2 の補数として解釈されるため、戻り値は -2147483648 ~ 2147483647 の整数になります。

オペランド

expression1 : Number - 数値。

expression2 : Number - 数値。

戻り値

Number - ビット演算の結果。

例

次の例では、数値のビット表現を比較し、同じ位置のビットが共に 1 であった場合にのみ 1 を返します。次の ActionScript コードでは、13 (2 進数 1101) と 11 (2 進数 1011) を加算し、両方とも 1 の桁について 1 を返します。

```
var insert:Number = 13;
var update:Number = 11;
trace(insert & update); // output : 9 (or 1001 binary)
```

2 つの数値のうち、共に 1 なのは先頭と末尾の桁だけであるため、13 と 11 の演算結果は 9 になります。

次の例は、戻り値の変換の動作を示しています。

```
trace(0xFFFFFFFF); // 4294967295
trace(0xFFFFFFFF & 0xFFFFFFFF); // -1
trace(0xFFFFFFFF & -1); // -1
trace(4294967295 & -1); // -1
trace(4294967295 & 4294967295); // -1
```

関連項目

[&= ビット単位の論理積 \(AND\) 代入演算子](#), [^ ビット単位の排他的論理和 \(XOR\) 演算子](#),

[^= ビット単位の排他的論理和 \(XOR\) 代入演算子](#), [| ビット単位の論理和 \(OR\) 演算子](#),

[|= ビット単位の排他的論理和 \(OR\) 代入演算子](#), [~ ビット単位の否定 \(NOT\) 演算子](#)

&= ビット単位の論理積 (AND) 代入演算子

expression1 &= expression2

expression1 に *expression1& expression2* の値を代入します。たとえば、次の 2 つの式は同じです。

```
x &= y;  
x = x & y;
```

オペランド

expression1 : Number - 数値。

expression2 : Number - 数値。

戻り値

Number - *expression1 & expression2* の値。

例

次の例では、*x* に値 9 が代入されます。

```
var x:Number = 15;  
var y:Number = 9;  
trace(x &= y); // output: 9
```

関連項目

[& ビット単位の論理積 \(AND\) 演算子](#), [^ ビット単位の排他的論理和 \(XOR\) 演算子](#), [^= ビット単位の排他的論理和 \(XOR\) 代入演算子](#), [| ビット単位の論理和 \(OR\) 演算子](#), [|= ビット単位の排他的論理和 \(OR\) 代入演算子](#), [~ ビット単位の否定 \(NOT\) 演算子](#)

<< ビット単位の左シフト演算子

expression1 << expression2

expression1 と *expression2* を 32 ビット整数に変換し、*expression2* の変換により生成された整数で指定される桁数だけ *expression1* 内のすべてのビットを左にシフトします。この演算の結果として空白になったビット位置には 0 が詰められ、左端からあふれたビットは破棄されます。値を 1 桁左にシフトすることは、2 を掛けることと同じ意味になります。

浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。正の整数は 4294967295 (0xFFFFFFFF) を最大値とする符号なし 16 進値に変換されます。最大値より大きい値は、32 ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2 の補数表現を使用して、-2147483648 (0x8000000000) を最小値とする符号なし 16 進値に変換されます。最小値よりも小さい値は、さらに高い精度で 2 の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの 2 の補数として解釈されるため、戻り値は -2147483648 ~ 2147483647 の整数になります。

オペランド

`expression1` : Number - 左にシフトされる数値または式。

`expression2` : Number - 0 ~ 31 の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

`x = 1 << 10` では、整数 1 を左に 10 ビットシフトします。この演算の結果は `x = 1024` です。10 進の 1 は 2 進の 1 です。10 左にシフトした 2 進の 1 は 10000000000 になります。それを 10 進に戻すと 1024 になります。`x = 7 << 8` では、整数 7 を左に 8 ビットシフトします。この演算の結果は `x = 1792` です。10 進の 7 は 2 進の 111 です。8 ビット左にシフトした 2 進の 111 は 11100000000 になります。それを 10 進に戻すと 1792 になります。次の例の出力結果を見ると、ビットが 2 つ分、左にシフトされていることがわかります。

```
// 2 binary == 0010
// 8 binary == 1000
trace(2 << 2); // output: 8
```

関連項目

[>>= ビット単位の右シフト後代入演算子](#), [>> ビット単位の右シフト演算子](#), [<<= ビット単位の左シフト後代入演算子](#), [>>> ビット単位の符号なし右シフト演算子](#), [>>>= ビット単位の符号なし右シフト後代入演算子](#)

<<= ビット単位の左シフト後代入演算子

`expression1 <<= expression2`

この演算子はビット単位の左シフト (<<=) 演算を行い、その内容を結果として `expression1` に格納します。次の 2 つの式は等価です。

`A <<= BA = (A << B)`

オペランド

`expression1` : Number - 左にシフトされる数値または式。

`expression2` : Number - 0 ~ 31 の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

次の例では、ビット単位での左シフト後代入 (`<<=`) 演算子を使用して、すべてのビットを1桁ずつ左にシフトしています。

```
var x:Number = 4;  
// shift all bits one slot to the left.  
x <<= 1;  
trace(x); // output: 8  
// 4 decimal = 0100 binary  
// 8 decimal = 1000 binary
```

関連項目

[<< ビット単位の左シフト演算子](#), [>>= ビット単位の右シフト後代入演算子](#), [>> ビット単位の右シフト演算子](#)

~ ビット単位の否定 (NOT) 演算子

`~expression`

1の補数演算子またはビット単位の補数演算子とも呼ばれます。*expression* を 32 ビット符号付き整数に変更し、ビット単位で 1 の補数を適用します。つまり、0 を保持するすべてのビットは 1 に設定され、1 を保持するすべてのビットは 0 に設定されます。結果は、符号付き 32 ビット整数です。

たとえば、`0x7777` という 16 進値は、2 進数では `0111011101110111` と表現されます。

この 16 進値をビット単位で符号反転 (`-0x7777`) すると、2 進数では `1000100010001000` になります。

これは、16 進数の `0x8888` に相当します。したがって、`-0x7777` は `0x8888` となります。

ビット単位演算子は、フラグビットを表現する場合に最もよく使用されます（ブール値をそれぞれ 1 ビットにパックすることができます）。

浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。正の整数は 4294967295 (`0xFFFFFFFF`) を最大値とする符号なし 16 進値に変換されます。最大値より大きい値は、32 ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2 の補数表現を使用して、-2147483648 (`0x80000000`) を最小値とする符号なし 16 進値に変換されます。最小値よりも小さい値は、さらに高い精度で 2 の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの 2 の補数として解釈されるため、戻り値は -2147483648 から 2147483647 の整数になります。

オペランド

expression : Number - 数値。

戻り値

Number - ビット演算の結果。

例

次に、フラグビットでビット単位の否定(NOT)(-)演算子を使用する例を示します。

```
var ReadOnlyFlag:Number = 0x0001; // defines bit 0 as the read-only flag
var flags:Number = 0;
trace(flags);
/* To set the read-only flag in the flags variable,
the following code uses the bitwise OR:
*/
flags |= ReadOnlyFlag;
trace(flags);
/* To clear the read-only flag in the flags variable,
first construct a mask by using bitwise NOT on ReadOnlyFlag.
In the mask, every bit is a 1 except for the read-only flag.
Then, use bitwise AND with the mask to clear the read-only flag.
The following code constructs the mask and performs the bitwise AND:
*/
flags &= ~ReadOnlyFlag;
trace(flags);
// output: 0 1 0
```

関連項目

[& ビット単位の論理積 \(AND\) 演算子](#), [&= ビット単位の論理積 \(AND\) 代入演算子](#), [^ ビット単位の排他的論理和 \(XOR\) 演算子](#), [^= ビット単位の排他的論理和 \(XOR\) 代入演算子](#), [| ビット単位の論理和 \(OR\) 演算子](#), [|= ビット単位の排他的論理和 \(OR\) 代入演算子](#)

| ビット単位の論理和 (OR) 演算子

expression1 | expression2

expression1とexpression2を32ビット符号なし整数に変換し、expression1とexpression2の対応ビットの少なくとも一方が1である各ビット位置に1を返します。浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。結果は、新しい32ビット整数です。

正の整数は4294967295(0xFFFFFFFF)を最大値とする符号なし16進値に変換されます。最大値より大きい値は、32ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2の補数表現を使用して、-2147483648(0x8000000000)を最小値とする符号なし16進値に変換されます。最小値よりも小さい値は、さらに高い精度で2の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの 2 の補数として解釈されるため、戻り値は -2147483648 ~ 2147483647 の整数になります。

オペランド

expression1 : Number - 数値。

expression2 : Number - 数値。

戻り値

Number - ビット演算の結果。

例

次に、ビット単位の論理和 (OR) (`|`) 演算の例を示します。

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 | 1001 = 1111
trace(x | y); // returns 15 decimal (1111 binary)
```

ビット単位の論理和 (`|`) と通常の論理和 (`||`) を混同しないようにしてください。

関連項目

& ビット単位の論理積 (AND) 演算子 , `&=` ビット単位の論理積 (AND) 代入演算子 , `^` ビット単位の排他的論理和 (XOR) 演算子 , `^=` ビット単位の排他的論理和 (XOR) 代入演算子 , `|=` ビット単位の排他的論理和 (OR) 代入演算子 , `~` ビット単位の否定 (NOT) 演算子

`|=` ビット単位の排他的論理和 (OR) 代入演算子

`expression1 |= expression2`

`expression1` に `expression1 | expression2` の値を代入します。たとえば、次の 2 つのステートメントは同じです。

```
x |= y; and x = x | y;
```

オペランド

expression1 : Number - 数値または変数。

expression2 : Number - 数値または変数。

戻り値

Number - ビット演算の結果。

例

次にビット単位の排他的論理和(OR)代入(|=)演算子の使用例を示します。

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 |= 1001 = 1111
trace(x |= y); // returns 15 decimal (1111 binary)
```

関連項目

& ビット単位の論理積(AND)演算子, &= ビット単位の論理積(AND)代入演算子, ^ ビット単位の排他的論理和(XOR)演算子, ^= ビット単位の排他的論理和(XOR)代入演算子, | ビット単位の論理和(OR)演算子, |= ビット単位の排他的論理和(OR)代入演算子, ~ ビット単位の否定(NOT)演算子

» ビット単位の右シフト演算子

expression1 >> *expression2*

*expression1*と*expression2*を32ビット整数に変換し、*expression2*の変換により生成された整数で指定される桁数だけ*expression1*内のすべてのビットを右にシフトします。シフト後、右端のビットは破棄されます。元の*expression*の符号を維持するには、*expression1*の最上位ビット(左端のビット)が0である場合は左側のビットに0を置き、最上位ビットが1である場合には1を置きます。値を右に1つシフトすることは、2で割って剰余を切り捨てることと同じです。

浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。正の整数は4294967295(0xFFFFFFFF)を最大値とする符号なし16進値に変換されます。最大値より大きい値は、32ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2の補数表現を使用して、-2147483648(0x8000000000)を最小値とする符号なし16進値に変換されます。最小値よりも小さい値は、さらに高い精度で2の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの2の補数として解釈されるため、戻り値は-2147483648～2147483647の整数になります。

オペランド

expression1 : Number - 右にシフトされる数値または式。

expression2 : Number - 0～31の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

次の例では、65535 を 32 ビット整数に変換し、右側に 8 ビットシフトします。

```
var x:Number = 65535 >> 8;  
trace(x); // outputs 255
```

次の例は、前の例の結果を示しています。

```
var x:Number = 255;
```

これは、10 進数の 65535 が 2 進数の 1111111111111111 (1 が 16 個) であり、8 ビットだけ右にシフトすると 2 進数の 11111111 になるためです。2 進数の 11111111 は 10 進数の 255 です。整数は 32 ビットであるため、最上位ビットの 0 で残りのビットを埋めます。

次の例では、-1 を 32 ビット整数に変換し、右側に 1 ビットシフトします。

```
var x:Number = -1 >> 1;  
trace(x); // outputs -1
```

次の例は、前の例の結果を示しています。

```
var x:Number = -1;
```

これは、10 進数の -1 が 2 進数の 11111111111111111111111111111111 (1 が 32 個) に等しく、右に 1 ビットシフトすると最下位ビット (右端のビット) が切り捨てられ、最上位ビットに 1 が詰められるためです。その結果は 2 進数の 11111111111111111111111111111111 (1 が 32 個) となり、これは 32 ビットの整数 -1 を表します。

関連項目

[>>= ビット単位の右シフト後代入演算子](#)

>>= ビット単位の右シフト後代入演算子

expression1 >>= *expression2*

この演算子はビット単位の右シフト演算を行い、その内容を結果として *expression1* に格納します。

次の 2 つのステートメントは等価です。

```
A >>= B; and A = (A >> B);
```

オペランド

expression1 : Number - 右にシフトされる数値または式。

expression2 : Number - 0 ~ 31 の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

次のコメント付きコードは、ビット単位の右シフト後代入 ($>>=$) 演算子の使用例です。

```
function convertToBinary(numberToConvert:Number):String {
    var result:String = "";
    for (var i = 0; i<32; i++) {
        // Extract least significant bit using bitwise AND
        var lsb:Number = numberToConvert & 1;
        // Add this bit to the result
        string result = (lsb ? "1" : "0")+result;
        // Shift numberToConvert right by one bit, to see next bit
        numberToConvert >>= 1;
    }
    return result;
}
trace(convertToBinary(479));
// Returns the string 0000000000000000000000111011111
// This string is the binary representation of the decimal
// number 479
```

関連項目

[>> ビット単位の右シフト演算子](#)

»» ビット単位の符号なし右シフト演算子

expression1 »» *expression2*

左側のビットには常に 0 が詰められるために元の *expression* の符号が保持されないという点を除いて、ビット単位の右シフト ($>>$) 演算子と同じです。

浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。正の整数は 4294967295 (0xFFFFFFFF) を最大値とする符号なし 16 進値に変換されます。最大値より大きい値は、32 ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2 の補数表現を使用して、-2147483648 (0x8000000000) を最小値とする符号なし 16 進値に変換されます。最小値よりも小さい値は、さらに高い精度で 2 の補数に変換された上で、最上位の桁が切り捨てられます。

オペランド

expression1 : Number - 右にシフトされる数値または式。

expression2 : Number - 0 ~ 31 の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

次の例では、-1を32ビット整数に変換し、右側に1ビットシフトします。

```
var x:Number = -1 >>> 1;  
trace(x); // output: 2147483647
```

これは、10進数の-1が2進数の111111111111111111111111111111(1が32個)であり、右に符号なしで1ビットシフトする場合は最下位(右端)ビットが切り捨てられ、最上位(左端)ビットに0が詰められるためです。その結果は2進数の011111111111111111111111111111となります。これは、32ビット整数の2147483647を表します。

関連項目

[>>= ビット単位の右シフト後代入演算子](#)

>>>= ビット単位の符号なし右シフト後代入演算子

expression1 >>>= *expression2*

ビット単位の符号なし右シフト演算を行い、その内容を結果として*expression1*に格納します。次の2つのステートメントは等価です。

```
A >>>= B; and A = (A >>> B);
```

オペランド

expression1 : Number - 右にシフトされる数値または式。

expression2 : Number - 0～31の整数に変換される数値または式。

戻り値

Number - ビット演算の結果。

例

関連項目

[>>> ビット単位の符号なし右シフト演算子](#), [>>= ビット単位の右シフト後代入演算子](#)

^ ビット単位の排他的論理和 (XOR) 演算子

expression1 ^ expression2

expression1 と *expression2* を 32 ビット符号なし整数に変換し、*expression1* と *expression2* の対応ビットのいずれか一方のみが 1 である各ビット位置に 1 を返します。浮動小数点数は、小数点以下が切り捨てられて整数に変換されます。結果は、新しい 32 ビット整数です。

正の整数は 4294967295 (0xFFFFFFFF) を最大値とする符号なし 16 進値に変換されます。最大値より大きい値は、32 ビットを超えないように、変換時に最上位の桁が切り捨てられます。負の値は、2 の補数表現を使用して、-2147483648 (0x8000000000) を最小値とする符号なし 16 進値に変換されます。最小値よりも小さい値は、さらに高い精度で 2 の補数に変換された上で、最上位の桁が切り捨てられます。

戻り値は符号付きの 2 の補数として解釈されるため、戻り値は -2147483648 ~ 2147483647 の整数になります。

オペランド

expression1 : Number - 数値。

expression2 : Number - 数値。

戻り値

Number - ビット演算の結果。

例

次の例では、ビット単位の排他的論理和 (XOR) 演算子を 10 進数である 15 および 9 に適用し、その結果を変数 *x* に代入します。

```
// 15 decimal = 1111 binary
// 9 decimal = 1001 binary
var x:Number = 15 ^ 9;
trace(x);
// 1111 ^ 1001 = 0110
// returns 6 decimal (0110 binary)
```

関連項目

[& ビット単位の論理積 \(AND\) 演算子](#), [&= ビット単位の論理積 \(AND\) 代入演算子](#), [^= ビット単位の排他的論理和 \(XOR\) 代入演算子](#), [| ビット単位の論理和 \(OR\) 演算子](#), [|= ビット単位の排他的論理和 \(OR\) 代入演算子](#), [~ ビット単位の否定 \(NOT\) 演算子](#)

$\wedge=$ ビット単位の排他的論理和 (XOR) 代入演算子

expression1 $\wedge=$ *expression2*

expression1 に *expression1* \wedge *expression2* の値を代入します。たとえば、次の 2 つのステートメントは同じです。

```
x  $\wedge=$  y x = x  $\wedge$  y
```

オペランド

expression1 : Number - 整数および変数。

expression2 : Number - 整数および変数。

戻り値

Number - ビット演算の結果。

例

次の例は、ビット単位の排他的論理和 (XOR) ($\wedge=$) 演算です。

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
trace(x  $\wedge=$  y); // returns 6 decimal (0110 binary)
```

関連項目

[& ビット単位の論理積 \(AND\) 演算子](#), [&= ビット単位の論理積 \(AND\) 代入演算子](#), [^ ビット単位の排他的論理和 \(XOR\) 演算子](#), [| ビット単位の論理和 \(OR\) 演算子](#), [|= ビット単位の排他的論理和 \(OR\) 代入演算子](#), [~ ビット単位の否定 \(NOT\) 演算子](#)

/* コメントブロック区切り記号演算子

```
/* comment */
/* comment
comment */
```

スクリプトコメントのブロックを示します。開始コメントタグ ($/*$) と終了コメントタグ ($*/$) の間の文字はすべてコメントと解釈され、ActionScript インタプリタでは無視されます。單一行のコメントを指定するには、 $//($ コメント行区切り記号) を使用します。連續した複数行のコメントブロックを指定するには、 $/*($ コメントブロック区切り記号) を使用します。この形式のコメントブロック区切り記号を使用する際に閉じるタグ ($*/$) を省略すると、エラーメッセージが返されます。コメントをネストしようとすると、エラーメッセージが返されます。コメントの終わりは、開始コメントタグ ($/*$) の後、最初に出現する終了コメントタグ ($*/$) によって示されます。開始コメントタグ ($/*$) がその間にいくつ指定されていても結果は同じです。

オペランド

comment - 任意の文字。

例

次のスクリプトでは、スクリプトの先頭にコメント区切り記号を使用します。

```
/* records the X and Y positions of  
the ball and bat movie clips */  
var ballX:Number = ball_mc._x;  
var ballY:Number = ball_mc._y;  
var batX:Number = bat_mc._x;  
var batY:Number = bat_mc._y;
```

次のように、コメントをネストしようとすると、エラーメッセージが表示されます。

```
/* this is an attempt to nest comments.  
/* But the first closing tag will be paired  
with the first opening tag */  
and this text will not be interpreted as a comment */
```

関連項目

[// コメント行区切り記号演算子](#)

, カンマ演算子

(*expression1* , *expression2* [, *expressionN...*])

式 *expression1*、式 *expression2*、... の順に評価します。通常、この演算子は `for` ループステートメントで使用します。括弧()演算子と組み合わせて使用することもあります。

オペランド

expression1 : Number - 評価される式。

expression2 : Number - 評価される式。

expressionN : Number - 上記に加えて評価される任意の個数の式。

戻り値

Object - *expression1*、*expression2*などの値。

例

次の例では、`for` ループでカンマ(,)演算子を使用しています。

```
for (i = 0, j = 0; i < 3 && j < 3; i++, j+=2) {  
    trace("i = " + i + ", j = " + j);  
}  
// Output:
```

```
// i = 0, j = 0  
// i = 1, j = 2
```

次の例では、括弧()演算子なしでカンマ(,)演算子を使用しています。括弧()演算子がない場合、カンマ演算子は最初の式の値だけを返していることがわかります。

```
var v:Number = 0;  
v = 4, 5, 6;  
trace(v); // output: 4
```

次の例では、括弧()演算子と組み合わせてカンマ(,)演算子を使用しています。括弧()演算子と組み合わせた場合、カンマ演算子は最後の式の値を返していることがわかります。

```
var v:Number = 0;  
v = (4, 5, 6);  
trace(v); // output: 6
```

次の例では、括弧()演算子なしでカンマ(,)演算子を使用しています。カンマ演算子はすべての式を順番に評価した上で最初の式の値を返していることがわかります。2番目の式 z++ では、z に1を加えています。

```
var v:Number = 0;  
var z:Number = 0;  
v = v + 4 , z++, v + 6;  
trace(v); // output: 4  
trace(z); // output: 1
```

次の例は、括弧()演算子が使われている以外は、前の例と同じです。括弧()演算子と組み合わせた場合、カンマ(,)演算子は一連の式の最後の式の値を返していることがわかります。

```
var v:Number = 0;  
var z:Number = 0;  
v = (v + 4, z++, v + 6);  
trace(v); // output: 6  
trace(z); // output: 1
```

関連項目

[\(\) 括弧演算子](#)

加算結合(ストリング)演算子

string1 add string2

非推奨 Flash Player 5 以降では使用しないでください。Flash Player 5 以降用のコンテンツを作成する場合は、加算(+)演算子を使用することをお勧めします。

メモ : Flash Lite 2.0 では、add 演算子も使用されなくなりました。代わりに加算(+)演算子を使用します。

複数のストリングを連結します。加算(+)演算子はFlash 4の連結(&)演算子に代わるもので、連結(&)演算子が使用されたFlash Player 4ファイルをFlash 5以降のオーサリング環境に読み込むと、ストリング連結に加算(+)演算子を使用するように自動的に変換されます。バージョン4以前のFlash Player用のコンテンツを作成する場合は、加算(+)演算子を使用して、ストリングを連結する必要があります。

オペランド

string1 : String - ストリング。

string2 : String - ストリング。

戻り値

String - 結合されたストリング。

関連項目

[+ 加算演算子](#)

？条件演算子

expression1 ? expression2 : expression3

*expression1*を評価し、*expression1*の値がtrueである場合は*expression2*の値を返します。それ以外の場合は*expression3*の値を返します。

オペランド

expression1 : Object - *expression1* - 評価結果がブール値になる式。通常はx < 5などの比較式。

expression2 : Object - 任意のタイプの値。

expression3 : Object - 任意のタイプの値。

戻り値

Object - *expression2*または*expression3*の値。

例

次のステートメントでは、*expression1*の評価結果がtrueなので、変数xの値が変数zに代入されます。

```
var x:Number = 5;
var y:Number = 10;
var z = (x < 6) ? x: y;
trace (z); // returns 5
```

次に、簡単な条件ステートメントの例を示します。

```
var timecode:String = (new Date().getHours() < 11) ? "AM" : "PM";
trace(timecode);
```

次のように、同じ条件ステートメントを、もう少し長く記述することもできます。

```
if (new Date().getHours() < 11) {
    var timecode:String = "AM";
} else {
    var timecode:String = "PM";
} trace(timecode);
```

-- デクリメント演算子

--*expression*
expression--

*expression*から1を引くプリデクリメント単項演算子またはポストデクリメント単項演算子です。*expression*は、変数、配列のエレメント、またはオブジェクトのプロパティです。プリデクリメント形式の演算子(--*expression*)は、*expression*から1を減算し、結果を返します。ポストデクリメント形式の演算子(*expression*--)は、*expression*から1を減算し、*expression*の初期値(減算前の値)を返します。

オペランド

expression : Number - 数値、または評価結果が数値になる変数。

戻り値

Number - デクリメントされた値の結果。

例

プリデクリメント形式の演算子は、xを2にデクリメント($x - 1 = 2$)して、結果をyとして返します。

```
var x:Number = 3;
var y:Number = --x; //y is equal to 2
```

ポストデクリメント形式の演算子は、xを2にデクリメント($x - 1 = 2$)して、xの元の値を結果yとして返します。

```
var x:Number = 3;
var y:Number = x--; //y is equal to 3
```

次の例は、10から1までループし、各ループでカウンタ変数iを1ずつ減らしています。

```
for (var i = 10; i>0; i--) {
    trace(i);
}
```

/ 除算演算子

expression1 / expression2

expression1 を *expression2* で除算します。除算演算の結果は倍精度の浮動小数です。

オペランド

expression : Number - 数値、または評価結果が数値になる変数。

戻り値

Number - 浮動小数の演算結果。

例

次のステートメントでは、ステージの現在の幅と高さを除算します。結果は [出力] パネルに表示されます。

```
trace(Stage.width/2);  
trace(Stage.height/2);
```

ステージの幅と高さが 550 x 400 (デフォルト) の場合、275 と 150 が出力されます。

関連項目

[% 剰余演算子](#)

/= 除算後代入演算子

expression1 /= expression2

expression1 に *expression1 / expression2* の値を代入します。たとえば、次の 2 つのステートメントは同じです。

```
x /= y; and x = x / y;
```

オペランド

expression1 : Number - 数値、または評価結果が数値になる変数。

expression2 : Number - 数値、または評価結果が数値になる変数。

戻り値

Number - 数値。

例

次のコードは、変数と数値を伴う除算後代入 (/=) 演算子の使用例です。

```
var x:Number = 10;  
var y:Number = 2;  
x /= y; trace(x); // output: 5
```

関連項目

/ 除算演算子

. ドット演算子

*object.property_or_method
instancename.variable
instancename.childinstance
instancename.childinstance.variable*

ネストされた子のムービークリップ、変数、またはプロパティにアクセスするためにムービークリップの階層をナビゲートする場合に使用します。ドット演算子は、オブジェクトまたはトップレベルクラスのプロパティを調べたり、プロパティを設定するときに使用します。また、オブジェクトまたはトップレベルクラスのメソッドを実行したり、データ構造体を作成する場合にも使用されます。

オペランド

object : Object - クラスのインスタンス。任意のビルトインクラスまたはカスタムクラスのインスタンスを指定できます。このパラメータは、常にドット(.)演算子の左側で使用します。

property_or_method - オブジェクトに関連するプロパティまたはメソッドの名前。ビルトインオブジェクトの有効なメソッドとプロパティは、そのクラスのメソッドとプロパティの一覧表に示されています。このパラメータは、常にドット(.)演算子の右側で使用します。

instancename : MovieClip - ムービークリップのインスタンス名。**variable** - ドット(.)演算子の左側にあるインスタンス名は、ムービークリップのタイムラインの変数も表すことができます。

childinstance : MovieClip - 別のムービークリップの子になっている(ネストされている)ムービークリップインスタンス。

戻り値

Object - ドットの右側に指定されたメソッド、プロパティ、またはムービークリップ。

例

次の例では、ムービークリップ person_mc 内の変数 hairColor の現在の値を調べます。

```
person_mc.hairColor
```

Flash 4 のオーサリング環境では、ドットシンタックスはサポートされません。ただし、Flash Player 4 用にパブリッシュされた Flash MX 2004 ファイルでは、ドット演算子を使用できます。前の例は、次に示す従来の Flash 4 シンタックスと同じです。

```
/person_mc:hairColor
```

次の例では、_root スコープ内に新しいムービークリップを作成します。次に、container_mc というムービークリップ内にテキストフィールドを作成しています。テキストフィールドの autoSize プロパティを true に設定し、現在の日付を設定しています。

```
this.createEmptyMovieClip("container_mc", this.getNextHighestDepth());
this.container_mc.createTextField("date_txt", this.getNextHighestDepth(), 0, 0,
    100, 22);
this.container_mc.date_txt.autoSize = true;
this.container_mc.date_txt.text = new Date();
```

ドット(.) 演算子は、SWF ファイル内のインスタンスを指定したり、インスタンスのプロパティや値を設定する場合に使用します。

== 等価演算子

expression1 == expression2

2つの式の等価性をテストします。式が等しい場合、結果は true です。

"等価" の定義は、パラメータのデータ型により異なります。

- 数値と布尔値は値により比較され、それらの値が同じであれば、等しいと見なされます。
- スtringing 式は、文字数が同じで、同じ文字で構成されている場合に、等しいと見なされます。
- オブジェクト、配列、および関数を表す変数は、参照により比較されます。2 つの変数が同じオブジェクト、配列、または関数を参照する場合、それらの変数は等価です。2 つの別個の配列は、エレメント数が同じである場合でも、等しいとは見なされません。

値による比較では、*expression1* と *expression2* のデータ型が異なる場合、ActionScript は *expression2* のデータ型を *expression1* と同じデータ型に変換することを試みます。

オペランド

expression1 : Object - 数値、ストリング、布尔値、変数、オブジェクト、配列、または関数。

expression2 : Object - 数値、ストリング、布尔値、変数、オブジェクト、配列、または関数。

戻り値

Boolean - 比較結果を表す布尔値。

例

次の例では、if ステートメントで等価(==)演算子を使用します。

```
var a:String = "David", b:String = "David";
if (a == b) {
    trace("David is David");
}
```

次の例では、混在するデータ型の比較演算の結果を示します。

```
var x:Number = 5;
var y:String = "5";
trace(x == y); // output: true
var x:String = "5";
var y:String = "66";
trace(x == y); // output: false
var x:String = "chris";
var y:String = "steve";
trace(x == y); // output: false
```

次に参照による比較の例を示します。1番目の例では、長さとエレメントの同じ 2つの配列を比較しています。この 2つの配列に対し、等価演算子は false を返します。これらの配列は一見、同じように見えますが、参照による比較で等価と見なされるためには、同じ配列を参照していることが必要です。2番目の例では、変数 firstArray と同じ配列を指し示す thirdArray という変数を作成しています。この 2つの配列に対しては、等価演算子が true を返します。2つの変数が同じ配列を参照しているからです。

```
var firstArray:Array = new Array("one", "two", "three");
var secondArray:Array = new Array("one", "two", "three");
trace(firstArray == secondArray);
// will output false
// Arrays are only considered equal
// if the variables refer to the same array.
var thirdArray:Array = firstArray;
trace(firstArray == thirdArray); // will output true
```

関連項目

[! 論理否定 \(NOT\) 演算子 , != 不等価演算子 , !== 厳密な不等価演算子 ,](#)
[&& 論理積 \(AND\) 演算子 , || 論理和 \(OR\) 演算子 , === 厳密な等価演算子](#)

eq 等価(ストリング)演算子

expression1 eq expression2

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに `==`(等価) 演算子を使用します。

2つの式が等しいかどうか比較し、*expression1* のストリング表現が *expression2* のストリング表現と等しい場合は `true` を返し、それ以外の場合は `false` を返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - 比較した結果。

関連項目

[== 等価演算子](#)

› より大きい演算子

expression1 > expression2

2つの式を比較し、*expression1* が *expression2* より大きいかどうかを判定します。大きい場合は `true` を返します。*expression1* が *expression2* より小さいか等しい場合は、`false` を返します。ストリング式はアルファベット順で評価されます。すべての大文字は小文字に先行します。

オペランド

expression1 : Object - 数値またはストリング。

expression2 : Object - 数値またはストリング。

戻り値

Boolean - 比較結果を表すブール値。

例

次の例では、より大きい(>)演算子を使用して、テキストフィールド score_txt の値が 90 より大きいかどうかを判定しています。

```
if (score_txt.text>90) {  
    trace("Congratulations, you win!");  
} else {  
    trace("sorry, try again");  
}
```

gt より大きい(ストリング)演算子

expression1 gt expression2

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに >(より大きい) 演算子を使用します。

expression1 のストリング表現を *expression2* のストリング表現と比較し、*expression1* が *expression2* より大きい場合は *true* を返します。それ以外の場合は *false* を返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - 比較結果を表すブール値。

関連項目

[> より大きい演算子](#)

>= より大きいか等しい演算子

expression1 >= expression2

2つの式を比較し、*expression1* が *expression2* より大きいか等しいか (*true*)、または *expression1* が *expression2* より小さいか (*false*) を判定します。

オペランド

expression1 : Object - ストリング、整数、または浮動小数。

expression2 : Object - ストリング、整数、または浮動小数。

戻り値

Boolean - 比較結果を表すブール値。

例

次の例では、より大きいか等しい(\geq)演算子を使用して、現在の時刻(時)が12以上であるかどうかを判定しています。

```
if (new Date().getHours() >= 12) {  
    trace("good afternoon");  
} else {  
    trace("good morning");  
}
```

ge 大きい、または等しい(ストリング)演算子

expression1 ge *expression2*

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに \geq (より大きい、または等しい)演算子を使用します。

expression1 のストリング表現を *expression2* のストリング表現と比較し、*expression1* が *expression2* より大きいか等しい場合は `true` を返します。それ以外の場合は `false` を返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - 比較した結果。

関連項目

[>= より大きいか等しい演算子](#)

++ インクリメント演算子

++expression

expression++

expression に1を加えるプリインクリメント単項演算子またはポストインクリメント単項演算子です。*expression* は、変数、配列のエレメント、またはオブジェクトのプロパティです。プリインクリメント形式の演算子 (*++expression*) は、*expression* に1を加算し、結果を返します。ポストインクリメント形式の演算子 (*expression++*) は、*expression* に1を加算し、*expression* の初期値(加算前の値)を返します。

プリインクリメント形式の演算子は、*x*を2にインクリメント($x + 1 = 2$)して、結果を*y*として返します。

```
var x:Number = 1;
var y:Number = ++x;
trace("x:"+x); //traces x:2
trace("y:"+y); //traces y:2
```

ポストインクリメント形式の演算子は、*x*を2にインクリメント($x + 1 = 2$)して、*x*の元の値を結果*y*として返します。

```
var x:Number = 1;
var y:Number = x++;
trace("x:"+x); //traces x:2
trace("y:"+y); //traces y:1
```

オペランド

expression : Number - 数値、または評価結果が数値になる変数。

戻り値

Number - インクリメントした結果。

例

次の例では、++をポストインクリメント演算子として使用し、whileループを5回実行します。

```
var i:Number = 0;
while (i++ < 5) {
    trace("this is execution " + i);
}
/* output:
   this is execution 1
   this is execution 2
   this is execution 3
   this is execution 4
   this is execution 5
*/
```

次の例では、++をプリインクリメント演算子として使用します。

```
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(++i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

次の例では、`++`をプリインクリメント演算子として使用します。

```
var a:Array = [];
for (var i = 1; i <= 10; ++i) {
    a.push(i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

このスクリプトを実行すると、`1,2,3,4,5,6,7,8,9,10`という結果が[出力]パネルに表示されます。次の例では、`while`ループで`++`をポストインクリメントの演算子として使用しています。

```
// using a while loop
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(i++);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

次の例では、`for`ループで`++`をポストインクリメント演算子として使用しています。

```
// using a for loop
var a:Array = new Array();
for (var i = 0; i < 10; i++) {
    a.push(i);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

このスクリプトを実行すると、次の結果が[出力]パネルに表示されます。

`0,1,2,3,4,5,6,7,8,9`

!= 不等価演算子

`expression1 != expression2`

等価(`==`)演算子の正反対が真であるかどうかをテストします。`expression1`が`expression2`に等しい場合、結果は`false`になります。厳密な等価(`==`)演算子と同様、等価の定義は比較対象のデータ型によって異なります。次にその例を示します。

- 数値、ストリング、およびブール値は、値により比較されます。
- オブジェクト、配列、および関数は、参照で比較されます。
- 変数はその型に応じて、値または参照で比較されます。

値による比較は、2つの式が同じ値を持つという、ごく一般的な視点から見た等価のことです。たとえば、値で比較した場合、`(2 + 3)`という式と`(1 + 4)`という式は等価になります。

参照による比較の場合、2つの式が等価と見なされるのは、両者が同じオブジェクト、配列、または関数を参照しているときだけです。オブジェクト、配列、または関数によって保持された値は比較の対象になりません。

値による比較では、*expression1* と *expression2* のデータ型が異なる場合、ActionScript は *expression2* のデータ型を *expression1* と同じデータ型に変換することを試みます。

オペランド

expression1 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

expression2 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

戻り値

Boolean - 比較結果を表すブール値。

例

次の例では、不等値 (*!=*) 演算子の結果を示します。

```
trace(5 != 8); // returns true
trace(5 != 5) //returns false
```

次の例では、if ステートメントでの不等値 (*!=*) 演算子の使い方を示します。

```
var a:String = "David";
var b:String = "Fool";
if (a != b) {
    trace("David is not a fool");
}
```

次の例では、2つの関数の参照による比較を示します。

```
var a:Function = function() { trace("foo"); };
var b:Function = function() { trace("foo"); };
a(); // foo
b(); // foo
trace(a != b); // true
a = b;
a(); // foo
b(); // foo
trace(a != b); // false
// trace statement output: foo foo true foo foo false
```

次の例では、2つの配列の参照による比較を示します。

```
var a:Array = [ 1, 2, 3 ];
var b:Array = [ 1, 2, 3 ];
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a!=b); // true
a = b;
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a != b); // false
// trace statement output: 1,2,3 1,2,3 true 1,2,3 1,2,3 false
```

関連項目

[! 論理否定 \(NOT\) 演算子](#), [!== 厳密な不等価演算子](#), [&& 論理積 \(AND\) 演算子](#), [|| 論理和 \(OR\) 演算子](#), [== 等価演算子](#), [==== 厳密な等価演算子](#)

<> 不等価演算子

expression1 <> expression2

非推奨 Flash Player 5 以降では使用しないでください。この演算子は使用されなくなりました。

[!= \(inequality\) 演算子](#)を使用することをお勧めします。

等価 (==) 演算子の正反対が真であるかどうかをテストします。*expression1* が *expression2* に等しい場合、結果は `false` になります。等価 (==) 演算子の場合と同様に、等価の定義は比較するデータ型により異なります。

- 数値、ストリング、およびブール値は、値により比較されます。
- オブジェクト、配列、および関数は、参照で比較されます。
- 変数はその型に応じて、値または参照で比較されます。

オペランド

expression1 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

expression2 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

戻り値

Boolean - 比較結果を表すブール値。

関連項目

[!= 不等価演算子](#)

instanceof 演算子

object instanceof classConstructor

object が *classConstructor* のインスタンスまたは *classConstructor* のサブクラスであるかどうかをテストします。*instanceof* 演算子は、プリミティブタイプをラッパーオブジェクトに変換しません。たとえば、次のコードは `true` を返します。

```
new String("Hello") instanceof String;  
一方、次のコードは false を返します。  
"Hello" instanceof String;
```

オペランド

object : Object - ActionScript オブジェクト。

classConstructor : Function - String や Date などの ActionScript コンストラクタ関数への参照。

戻り値

Boolean - object が classConstructor のインスタンスまたはサブクラスである場合、instanceof は true を返し、そうでない場合は false を返します。また、_global instanceof Object は false を返します。

関連項目

[typeof 演算子](#)

< より小さい演算子

expression1 < expression2

2つの式を比較し、*expression1*が*expression2*より小さいかどうかを判定します。小さい場合は true を返します。*expression1*が*expression2*よりも大きいか等しい場合は、false を返します。ストリング式はアルファベット順で評価されます。すべての大文字は小文字に先行します。

オペランド

expression1 : Number - 数値またはストリング。

expression2 : Number - 数値またはストリング。

戻り値

Boolean - 比較結果を表すブール値。

例

次の例では、数値とストリングの両方について true と false が返される場合を示します。

```
trace(3 < 10); // true
trace(10 < 3); // false
trace("Allen" < "Jack"); // true
trace("Jack" < "Allen"); // false
trace("11" < "3"); // true
trace("11" < 3); // false (numeric comparison)
trace("C" < "abc"); // true
trace("A" < "a"); // true
```

lt より小さい(ストリング)演算子

expression1 lt expression2

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <(より小さい) 演算子を使用します。

expression1 と *expression2* を比較して、*expression1* が *expression2* よりも小さい場合は true を返し、そうでない場合は false を返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - 比較した結果。

関連項目

[< より小さい演算子](#)

<= より小さいか等しい演算子

expression1 <= expression2

2つの式を比較し、*expression1* が *expression2* よりも小さいか等しいかどうかを判定します。小さいか等しい場合は true を返します。*expression1* が *expression2* よりも大きい場合は、false を返します。ストリング式はアルファベット順で評価されます。すべての大文字は小文字に先行します。

オペランド

expression1 : Object - 数値またはストリング。

expression2 : Object - 数値またはストリング。

戻り値

Boolean - 比較結果を表すブール値。

例

次に、数値とストリングの両方について true と false が返される場合の例を示します。

```
trace(5 <= 10); // true
trace(2 <= 2); // true
trace(10 <= 3); // false
trace("Allen" <= "Jack"); // true
trace("Jack" <= "Allen"); // false
trace("11" <= "3"); // true
trace("11" <= 3); // false (numeric comparison)
trace("C" <= "abc"); // true
trace("A" <= a); // true
```

le 小さい、または等しい(ストリング)演算子

expression1 le expression2

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに `<=`(より小さい、または等しい) 演算子を使用します。

expression1 を *expression2* と比較し、*expression1* が *expression2* よりも小さいか等しい場合は `true` を返し、それ以外の場合は `false` を返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - 比較した結果。

関連項目

[<= より小さいか等しい演算子](#)

// コメント行区切り記号演算子

// comment

スクリプトコメントの先頭を示します。コメント行区切り記号 (`//`) と行末の間に表示される文字はすべてコメントと解釈され、ActionScript インタプリタで無視されます。

オペランド

comment - 任意の文字。

例

次のスクリプトは、コメント行区切り記号を使用して1行目、3行目、5行目、7行目をコメントとして識別します。

```
// record the X position of the ball movie clip
var ballX:Number = ball_mc._x;
// record the Y position of the ball movie clip
var ballY:Number = ball_mc._y;
// record the X position of the bat movie clip
var batX:Number = bat_mc._x;
// record the Y position of the ball movie clip
var batY:Number = bat_mc._y;
```

関連項目

[/* コメントブロック区切り記号演算子](#)

&& 論理積 (AND) 演算子

expression1 && expression2

オペランドをブール(論理)値として評価し、論理演算を行います。オペランドをブール(論理)値として評価し、論理演算を行います。オペランドがブール値を返す論理式(等価や非等価、比較などを調べる式)であれば、両式が共に `false` の場合のみ `false` を返します。それ以外の場合、つまりどちらか一方の式または両式が `true` であれば、戻り値は `false` になります。`expression1` の評価が `true` であれば、`expression2`(演算子の右側の式)が評価されます。`expression2` の評価が `true` であれば、最終結果は `true` です。そうでなければ、`false` です。式の評価は、`true&&true` は `true`、`true&&false` は `false`、`false&&false` は `false`、`false&&true` は `false` になります。

オペランド

`expression1` : Number - ブール値、またはブール値に変換される式。

`expression2` : Number - ブール値、またはブール値に変換される式。

戻り値

Boolean - 論理演算結果を表すブール値。

例

次の例では、論理積 (AND) (`&&`) 演算子を使用してゲームの勝敗判定をテストしています。`turns` 変数と `score` 変数は、ゲームの回数または得点に応じて更新されます。次のスクリプトでは、3回以内に 75 点以上を得点すると、[出力] パネルに "You Win the Game!" と表示されます。

```
var turns:Number = 2;
var score:Number = 77;
if ((turns <= 3) && (score >= 75)) {
```

```
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
// output: You Win the Game!
```

関連項目

[! 論理否定 \(NOT\) 演算子](#), [!= 不等価演算子](#), [!== 厳密な不等価演算子](#),
[|| 論理和 \(OR\) 演算子](#), [== 等価演算子](#), [=== 厳密な等価演算子](#)

and 論理積 (AND) 演算子

condition1 and condition2

非推奨 Flash Player 5 以降では使用しないでください。論理積 (`&&`) 演算子を使用することをお勧めします。

Flash Player 4 で論理積 (AND) (`&&`) 演算を実行します。両式の評価値が `true` である場合に、式全体が `true` になります。

オペランド

condition1 : Boolean - *condition1, condition2* `true` または `false` に評価される条件または式。

condition2 : Boolean - *condition1, condition2* `true` または `false` に評価される条件または式。

戻り値

Boolean - 論理演算結果を表すブール値。

関連項目

[&& 論理積 \(AND\) 演算子](#)

! 論理否定 (NOT) 演算子

! expression

変数や式のブール値を反転します。*expression* が変数で、その絶対値または変換された値が `true` である場合、`!expression` の値は `false` になります。式 `x && y` の評価が `false` である場合、式 `!(x && y)` の評価は `true` です。

次の式は、論理否定 (!) 演算子を使用した結果を示します。

`! true` は `false` を返し、`! false` は `true` を返します。

オペランド

expression : Boolean - 評価結果がブール値になる式または変数。

戻り値

Boolean - 論理演算結果を表すブール値。

例

次の例では、変数 happy が false に設定されています。if ステートメントの条件部 !happy を評価し、その結果が true である場合、trace() ステートメントによりストリングが[出力]パネルに表示されます。

```
var happy:Boolean = false;
if (!happy) {
    trace("don't worry, be happy"); //traces don't worry, be happy
}
```

false は true と等価であるため trace が実行されます。

関連項目

[!= 不等価演算子](#), [!== 厳密な不等価演算子](#), [&& 論理積（AND）演算子](#),
[|| 論理和（OR）演算子](#), [== 等価演算子](#), [=== 厳密な等価演算子](#)

not 論理否定 (NOT) 演算子

not expression

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに ! (logical NOT) 演算子を使用します。

Flash Player 4 で論理否定 (NOT) (!) 演算を行います。

オペランド

expression : Object - ブール値に変換される変数または式。

戻り値

Boolean - 論理演算の結果。

関連項目

[! 論理否定（NOT）演算子](#)

|| 論理和 (OR) 演算子

expression1 || expression2

オペランドをブール(論理)値として評価し、論理演算を行います。オペランドがブール値を返す論理式(等価や非等価、比較などを調べる式)であれば、両式が共に false の場合のみ false を返します。それ以外の場合、つまりどちらか一方の式または両式が true であれば、戻り値は true になります。

expression1 の評価が false であれば、*expression2*(演算子の右側の式)が評価されます。

expression2 の評価が false であれば、最終結果は false です。そうでなければ、true です。

expression1 の評価が true の場合、*expression2*に関数呼び出しが指定されていても、その関数は実行されません。

一方の式または両式の評価が true である場合、結果は true になります。両式の評価が false である場合のみ、結果は false になります。論理和 (OR) 演算子は任意の数のオペランドに適用できます。いずれかのオペランドの評価が true であれば、結果は true です。

オペランド

expression1 : Number - ブール値、またはブール値に変換される式。

expression2 : Number - ブール値、またはブール値に変換される式。

戻り値

Boolean - 論理演算の結果。

例

次の例では、if ステートメントで論理和 (OR)(||) 演算子を使用しています。2 番目の式の評価結果が true のので、最終結果は true になります。

```
var x:Number = 10;
var y:Number = 250;
var start:Boolean = false;
if ((x > 25) || (y > 200) || (start)) {
    trace("the logical OR test passed"); // output: the logical OR test passed
}
```

if ステートメントの条件の1つが true ($y > 200$) ので、"the logical OR test passed" というメッセージが表示されます。それ以外の2つの式が false と評価されても、true と評価される条件が1つでもあれば、if ブロックが実行されます。

次のように、*expression2*に関数呼び出しを指定すると、予期しない結果が生じる場合があります。演算子の左側の式が true に評価された場合、右側の式は評価されずに（関数 *fx2()* は呼び出されない）、左側の式の結果だけが返されます。

```
function fx1():Boolean {
    trace("fx1 called");
    return true;
}
function fx2():Boolean {
    trace("fx2 called");
    return true;
}
if (fx1() || fx2()) {
    trace("IF statement entered");
}
/* The following is sent to the Output panel: /* The following is sent to the log
   file: fx1 called IF statement entered */
```

関連項目

[！ 論理否定（NOT）演算子](#) , [!= 不等値演算子](#) , [!== 厳密な不等値演算子](#) ,
[&& 論理積（AND）演算子](#) , [== 等値演算子](#) , [==== 厳密な等値演算子](#)

or 論理和（OR）演算子

condition1 or *condition2*

非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに ||（論理和）演算子を使用します。

condition1 と *condition2* を評価し、いずれかの式が true であれば、式全体が true になります。

オペランド

condition1 : Boolean - 評価結果が true または false になる式。

condition2 : Boolean - 評価結果が true または false になる式。

戻り値

Boolean - 論理演算の結果。

関連項目

[|| 論理和（OR）演算子](#) , [| ビット単位の論理和（OR）演算子](#)

% 剰余演算子

expression1 % expression2

*expression1*を*expression2*で割ったときの剰余を計算します。*expression*パラメータがいずれも非数値である場合、剰余(%)演算子はパラメータを数値に変換しようとします。*expression*は数値、または数値に変換されるストリングです。

剰余演算結果の符号は、被除数(最初の数値)の符号と一致します。たとえば、`-4 % 3`と`-4 % -3`の評価結果は共に`-1`になります。

オペランド

expression1 : Number - 数値、または評価結果が数値になる式。

expression2 : Number - 数値、または評価結果が数値になる式。

戻り値

Number - 算術演算の結果。

例

次に、数値に対して剰余(%)演算子を使用する例を示します。

```
trace(12%5); // traces 2
trace(4.3%2.1); // traces 0.0999999999999996
trace(4%4); // traces 0
```

剰余(%)演算子は余りだけを返すため、1つ目のtraceステートメントでは`12/5`や`2.4`ではなく`2`が返されます。ところが、2番目のtraceステートメントでは`0.1`ではなく、`0.0999999999999996`が返されます。これは2進数計算では浮動小数点の精度に限度があるからです。

関連項目

/ [除算演算子](#), [round \(Math.round メソッド\)](#)

%= 剰余代入演算子

expression1 %= expression2

*expression1*に*expression1 % expression2*の値を代入します。次の2つのステートメントは等価です。

```
x %= y; and x = x % y;
```

オペランド

expression1 : Number - 数値、または評価結果が数値になる式。

expression2 : Number - 数値、または評価結果が数値になる式。

戻り値

Number - 算術演算の結果。

例

次の例では、値 4 を変数 x に代入します。

```
var x:Number = 14;  
var y:Number = 5;  
trace(x % y); // output: 4
```

関連項目

% 剰余演算子

* 乗算演算子

*expression1 * expression2*

2つの数値や式を乗算します。両方の式が整数であれば、その積は整数です。いずれかの式または両方の式が浮動小数であれば、その積は浮動小数になります。

オペランド

expression1 : Number - 数値、または評価結果が数値になる式。

expression2 : Number - 数値、または評価結果が数値になる式。

戻り値

Number - 整数または浮動小数。

例

シンタックス1: 次のステートメントは、整数 2 と 3 を乗算します。

```
trace(2*3); // output: 6
```

結果は、整数の 6 です。シンタックス2: 次のステートメントは、浮動小数 2.0 と 3.1416 を乗算します。

```
trace(2.0 * 3.1416); // output: 6.2832
```

結果は、浮動小数の 6.2832 です。

*= 乗算後代入演算子

```
expression1 *= expression2
```

expression1 に *expression1 * expression2* の値を代入します。たとえば、次の 2 つの式は同じです。

```
x *= y x = x * y
```

オペランド

expression1 : Number - 数値、または評価結果が数値になる式。

expression2 : Number - 数値、または評価結果が数値になる式。

戻り値

Number - *expression1 * expression2* の値。式を数値に変換できない場合は、NaN(非数) を返します。

例

シンタックス1: 次の例では、値 50 を変数 x に代入します。

```
var x:Number = 5;
var y:Number = 10;
trace(x *= y); // output: 50
```

シンタックス2: 次の例の 2 行目と 3 行目は、等号の右側の式を計算し、その結果を x と y にそれぞれ代入します。

```
var i:Number = 5;
var x:Number = 4 - 6;
var y:Number = i + 2;
trace(x *= y); // output: -14
```

関連項目

[* 乗算演算子](#)

new 演算子

```
new constructor()
```

新しい匿名のオブジェクトを作成し、constructor パラメータで指定された関数を呼び出します。new 演算子は、括弧内のオプションのパラメータと、キーワード this で参照される新しく作成されたオブジェクトを関数に渡します。constructor 関数は this を使用してオブジェクトの変数を設定できます。

オペランド

constructor : Object - 括弧で囲まれたオプションのパラメータが後に続く関数。この関数は通常、作成するオブジェクトのタイプの名前(Array、Number、Objectなど)です。

例

次の例では、Book()関数を作成し、new演算子を使用してbook1オブジェクトとbook2オブジェクトを作成します。

```
function Book(name, price){  
    this.name = name;  
    this.price = price;  
}  
  
book1 = new Book("Confederacy of Dunces", 19.95);  
book2 = new Book("The Floating Opera", 10.95);
```

次の例では、new演算子を使用して、18個のエレメントを含むArrayオブジェクトを作成します。

```
golfCourse_array = new Array(18);
```

関連項目

[\[\] 配列アクセス演算子](#), [{} オブジェクト初期化演算子](#)

ne 不等価(ストリング)演算子

expression1 ne expression2

非推奨 Flash Player 5以降では使用しないでください。この演算子の代わりに != (inequality) 演算子を使用します。

*expression1*を*expression2*と比較して、*expression1*が*expression2*と等しくない場合はtrueを返します。それ以外の場合はfalseを返します。

オペランド

expression1 : Object - 数値、ストリング、または変数。

expression2 : Object - 数値、ストリング、または変数。

戻り値

Boolean - *expression1*が*expression2*と等しくない場合はtrueを返します。それ以外の場合はfalseを返します。

関連項目

[!= 不等価演算子](#)

{ } オブジェクト初期化演算子

```
object = { name1 : value1 , name2 : value2 ,... nameN : valueN }
(expression1; [...expressionN])
```

新しいオブジェクトを作成し、指定された *name* と *value* プロパティペアで初期化します。この演算子を使用することは、`new Object` シンタックスを使用して代入演算子でプロパティペアを設定するのと同じです。新しく作成されるオブジェクトのプロトタイプとして、汎用の `Object` オブジェクトがあります。

また、フローを制御するステートメント (`for`、`while`、`if`、`else`、`switch`) や関数で、ひとかたまりのコードブロックであることを示すときにも、この演算子が使用されます。

オペランド

`object` : `Object` - 作成するオブジェクト。*name1,2,...N* プロパティの名前。*value1,2,...N* 各 *name* プロパティに対応する値。

戻り値

`Object` -

シンタックス1: `Object` オブジェクト

シンタックス2: なし。ただし、関数で明示的に `return` ステートメントが指定されている場合は、その関数で戻り値のタイプが指定されます。

例

次のコードの1行目はオブジェクト初期化 ({ }) 演算子を使用して空のオブジェクトを作成し、2行目はコンストラクタ関数を使用して新しいオブジェクトを作成します。

```
var object:Object = {};
var object:Object = new Object();
```

次の例では、オブジェクト `account` を作成し、`name`、`address`、`city`、`state`、`zip`、`balance` の各プロパティを対応する値で初期化します。

```
var account:Object = {name:"Macromedia, Inc.", address:"600 Townsend Street",
    city:"San Francisco", state:"California", zip:"94103", balance:"1000"};
for (i in account) {
    trace("account." + i + " = " + account[i]);
}
```

次の例では、配列とオブジェクトの初期化演算子を相互にネストする方法を示します。

```
var person:Object = {name:"Gina Vechio", children:["Ruby", "Chickie", "Puppa"]};
```

次の例では、前の例と同じ内容を、コンストラクタ関数で生成します。

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array();
person.children[0] = "Ruby";
person.children[1] = "Chickie";
person.children[2] = "Puppa";
```

上の例で示した ActionScript は、次のように記述することもできます。

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array("Ruby", "Chickie", "Puppa");
```

関連項目

[Object](#)

() 括弧演算子

(*expression1 [, expression2]*)
(*expression1, expression2*)
function (*parameter1,..., parameterN*)

パラメータに対してグループ化演算を実行するか、複数の式を順番に評価します。または、パラメータを囲み、結果をパラメータとして括弧の外側にある関数に渡します。

シンタックス1: 式内での演算子の実行順序を制御します。括弧は通常の優先順位を無効にし、括弧内の式を最初に評価します。括弧がネストされている場合は、最も内側の括弧から順に外側の括弧へと内容が評価されます。

シンタックス2: カンマ区切りの一連の式を順番に評価し、最後に実行された式の結果を返します。

シンタックス3: パラメータを囲み、パラメータとして括弧の外側にある関数に渡します。

オペランド

expression1 : Object - 数値、ストリング、変数、またはテキスト。

expression2 : Object - 数値、ストリング、変数、またはテキスト。

function : Function - 括弧の内容に対して実行される関数。

parameter1...parameterN : Object - これらのパラメータの実行結果がパラメータとして括弧の外側の関数に渡されます。

例

シンタックス1: 次のステートメントは、括弧を使用して式の実行順序を制御しています(各式の値が[出力]パネルに表示されます)。

```
trace((2 + 3)*(4 + 5)); // displays 45 trace((2 + 3) * (4 + 5)); // writes 45
trace(2 + (3 * (4 + 5))); // // displays 29 trace(2 + (3 * (4 + 5))); // //
writes 29 trace(2+(3*4)+5); // displays 19 trace(2 + (3 * 4) + 5); // writes 19
```

シンタックス2: 次の例は、関数 foo() を評価した後に、関数 bar() を評価し、その後、式 a + b の結果を返します。

```
var a:Number = 1;
var b:Number = 2;
function foo() { a += b; }
function bar() { b *= 10; }
trace((foo(), bar(), a + b)); // outputs 23
```

シンタックス3: 次の例は、関数で括弧を使用する場合を示しています。

```
var today:Date = new Date();
trace(today.getFullYear()); // traces current year
function traceParameter(param):Void { trace(param); }
traceParameter(2 * 2); //traces 4
```

関連項目

[with ステートメント](#)

== 厳密な等価演算子

expression1 == expression2

2つの式が等しいかどうかをテストします。厳密な等価(==)演算子は、データ型が変換されない点を除いては、等価(==)演算子と同じです。両方の式が、そのデータ型も含めて等しい場合、結果は true です。

"等価" の定義は、パラメータのデータ型により異なります。

- 数値とブール値は値により比較され、それらの値が同じであれば、等しいと見なされます。
- スtringing式は、文字数が同じで、同じ文字で構成されている場合に、等しいと見なされます。
- オブジェクト、配列、および関数を表す変数は、参照により比較されます。2つの変数が同じオブジェクト、配列、または関数を参照する場合、それらの変数は等価です。2つの別個の配列は、エレメント数が同じである場合でも、等しいとは見なされません。

オペランド

expression1 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

expression2 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

戻り値

Boolean - 比較結果を表すブール値。

例

次のコードのコメントは、等価演算子および厳密な等価演算子を使用した演算の戻り値を示しています。

```
// Both return true because no conversion is done
var string1:String = "5";
var string2:String = "5";
trace(string1 == string2); // true
trace(string1 === string2); // true
// Automatic data typing in this example converts 5 to "5"
var string1:String = "5";
var num:Number = 5;
trace(string1 == num); // true
trace(string1 === num); // false
// Automatic data typing in this example converts true to "1"
var string1:String = "1";
var booll:Boolean = true;
trace(string1 == booll); // true
trace(string1 === booll); // false
// Automatic data typing in this example converts false to "0"
var string1:String = "0";
var bool2:Boolean = false;
trace(string1 == bool2); // true
trace(string1 === bool2); // false
```

次の例では、厳密な等価で比較する場合に、参照を格納する変数とリテラル値を格納する変数の扱いがどのように違うかを示しています。この結果を見ると、String クラスでは new 演算子の使用を避け、ストリングリテラルを使用する理由が明らかになります。

```
// Create a string variable using a literal value
var str:String = "asdf";
// Create a variable that is a reference
var stringRef:String = new String("asdf");
// The equality operator does not distinguish among literals, variables,
// and references
trace(stringRef == "asdf"); // true
trace(stringRef == str); // true
trace("asdf" == str); // true
// The strict equality operator considers variables that are references
// distinct from literals and variables
trace(stringRef === "asdf"); // false
trace(stringRef === str); // false
```

関連項目

[! 論理否定 \(NOT\) 演算子](#), [!= 不等価演算子](#), [!== 厳密な不等価演算子](#),
[&& 論理積 \(AND\) 演算子](#), [|| 論理和 \(OR\) 演算子](#), [== 等価演算子](#)

!== 厳密な不等価演算子

expression1 !== expression2

厳密な等価(==)演算子の正反対が真であるかどうかをテストします。厳密な不等価演算子の動作は、データ型が変換されない点を除いて、不等価演算子と同じです。

*expression1*が*expression2*と等しく、両者のデータ型が同じである場合、結果は false になります。厳密な等価(==)演算子と同様、等価の定義は比較対象のデータ型によって異なります。次にその例を示します。

- 数値、ストリング、およびブール値は、値により比較されます。
- オブジェクト、配列、および関数は、参照で比較されます。
- 変数はその型に応じて、値または参照で比較されます。

オペランド

expression1 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

expression2 : Object - 数値、ストリング、ブール値、変数、オブジェクト、配列、または関数。

戻り値

Boolean - 比較結果を表すブール値。

例

次のコードのコメントは、等価演算子(==)、厳密な等価演算子(===)、厳密な不等価演算子(!==)を使用した演算の戻り値を示しています。

```
var s1:String = "5";
var s2:String = "5";
var s3:String = "Hello";
var n:Number = 5;
var b:Boolean = true;
trace(s1 == s2); // true
trace(s1 == s3); // false
trace(s1 === n); // true
trace(s1 == b); // false
trace(s1 === s2); // true
trace(s1 === s3); // false
trace(s1 === n); // false
trace(s1 === b); // false
trace(s1 !== s2); // false
trace(s1 !== s3); // true
trace(s1 !== n); // true
trace(s1 !== b); // true
```

関連項目

[! 論理否定 \(NOT\) 演算子](#), [!= 不等価演算子](#), [&& 論理積 \(AND\) 演算子](#), [|| 論理和 \(OR\) 演算子](#), [== 等価演算子](#), [==== 厳密な等価演算子](#)

" ストリング区切り記号演算子

`"text"`

引用符 ("") で前後を囲んだ文字はリテラル値を表します。変数や数値、その他の ActionScript エレメントではなく、ストリングと見なされます。

オペランド

`text : String` - 0 個以上の文字のシーケンス。

例

次の例では、引用符 ("") を使用して、変数 `yourGuess` の値がリテラルストリング "Prince Edward Island" であり、変数名ではないことを指定します。`province` の値は変数であり、リテラルではありません。`province` の値を決定するには、`yourGuess` の値を特定する必要があります。

```
var yourGuess:String = "Prince Edward Island";
submit_btn.onRelease = function() { trace(yourGuess); };
// displays Prince Edward Island in the Output panel
// writes Prince Edward Island to the log file
```

関連項目

[String, String 関数](#)

- 減算演算子

(Negation) `-expression`

(Subtraction) `expression1 - expression2`

符号反転や減算に使用します。

シンタックス1: 符号反転に使用する場合、数値 `expression` の符号を逆にします。シンタックス2: 減算に使用する場合、2つの式に対して算術的な減算を行い、`expression1` から `expression2` を減算します。両方の式が整数であれば、その差は整数です。いずれかの式または両方の式が浮動小数であれば、その差は浮動小数です。

オペランド

`expression1 : Number` - 数値、または評価結果が数値になる式。

`expression2 : Number` - 数値、または評価結果が数値になる式。

戻り値

Number - 整数または浮動小数。

例

シンタックス1: 次のステートメントは、式 $2 + 3$ の符号を逆にします。

```
trace(-(2+3)); // output: -5
```

シンタックス2: 次のステートメントは、整数 5 から整数 2 を減算します。

```
trace(5-2); // output: 3
```

結果は、整数の 3 です。シンタックス3: 次のステートメントは、浮動小数点数 3.25 から浮動小数点数 1.5 を減算します。

```
trace(3.25-1.5); // output: 1.75
```

結果は、浮動小数の 1.75 です。

-= 減算後代入演算子

expression1 -= *expression2*

expression1 に *expression1* - *expression2* の値を代入します。たとえば、次の2つのステートメントは同じです。

```
x -= y; x = x - y;
```

ストリング式は数値に変換される必要があります。変換されない場合は、NaN(非数)が返されます。

オペランド

expression1 : Number - 数値、または評価結果が数値になる式。

expression2 : Number - 数値、または評価結果が数値になる式。

戻り値

Number - 算術演算の結果。

例

次の例では、減算後代入(-=)演算子を使用して、 5 から 10 を減算し、その結果を変数 *x* に代入します。

```
var x:Number = 5;
var y:Number = 10;
x -= y; trace(x); // output: -5
```

次の例では、ストリングがどのように数値に変換されるかを示します。

```
var x:String = "5";
var y:String = "10";
x -= y; trace(x); // output: -5
```

関連項目

- [減算演算子](#)

: タイプ演算子

```
[ modifiers ] var variableName : type  
function functionName () : type { ... }  
function functionName ( parameter1:type , ... , parameterN:type ) [ :type ]{ ... }
```

厳密な型指定を行う際に使用します。この演算子では、変数のタイプ、関数の戻り値のタイプ、または関数パラメータのタイプを指定します。変数の宣言または代入で使用する場合、この演算子は変数のタイプを指定します。関数の宣言または定義で使用する場合、この演算子は関数の戻り値のタイプを指定します。関数定義の関数パラメータで使用する場合、この演算子はそのパラメータに指定する変数のタイプを指定します。

タイプはコンパイル時のみの機能です。すべてのタイプはコンパイル時にチェックされ、不一致が見つかった場合はエラーが報告されます。タイプの不一致は、代入操作、関数の呼び出し、ドット(.)演算子を使用したクラスメンバーの逆参照の際に発生する可能性があります。タイプの不一致エラーを防ぐには、厳密な型指定を使用します。

使用できるタイプとしては、すべてのネイティブオブジェクトタイプ、自分で定義したクラスとインターフェイス、Function および Void があります。認識されるネイティブタイプには、布尔(Boolean)、数值(Number)、ストリング(String) があります。また、すべてのビルトインクラスは、ネイティブタイプとしてサポートされます。

オペランド

variableName : Object - 変数の識別子。type ネイティブのデータ型、定義済みのクラス名、またはインターフェイス名。functionName 関数の識別子。parameter 関数の識別子。

例

シンタックス 1: 次の例では、userName という名前のパブリック変数を宣言します。この変数は String タイプです。この変数に空のストリングを代入します。

```
var userName:String = "";
```

シンタックス 2: 次の例は、関数パラメータのタイプを指定する方法を示しています。ここでは、integer という Number タイプのパラメータを受け取る randomInt() という関数を定義しています。

```
function randomInt(integer:Number):Number {  
    return Math.round(Math.random()*integer);  
}  
trace(randomInt(8));
```

シンタックス 3:次の例は、`squareRoot()` という名前の関数を定義します。この関数は `val` という Number タイプのパラメータを受け取り、`val` の平方根を返します。この戻り値も Number タイプです。

```
function squareRoot(val:Number):Number {  
    return Math.sqrt(val);  
}  
trace(squareRoot(121));
```

関連項目

[var ステートメント](#) , [function ステートメント](#)

typeof 演算子

`typeof(expression)`

`typeof` 演算子は `expression` を評価し、その式が String、MovieClip、Object、Function、Number、Boolean のいずれの値であるかを示すストリングを返します。

オペランド

`expression` : Object - ストリング、ムービークリップ、ボタン、オブジェクト、または関数。

戻り値

String - 式のタイプのストリング表現。次の表に、`typeof` 演算子を使用した結果を `expression` のタイプ別に示します。

expression のタイプ	結果
ストリング	string
ムービークリップ	movieclip
ボタン	object
テキストフィールド	object
数値	number
ブール値	boolean
オブジェクト	object
関数	function

関連項目

[instanceof 演算子](#)

void 演算子

`void expression`

void 演算子は式を評価した後、その値を破棄して、`undefined` を返します。void 演算子は、多くの場合、`==` 演算子を使用して `undefined` 値をテストする比較において使用します。

オペランド

`expression` : Object - 評価される式。

ステートメント

ステートメントとは、特定のアクションを実行または指定する言語エレメントです。たとえば、`return` ステートメントは、関数の実行結果を値として返します。`if` ステートメントは、次に行うべき処理を判定するための条件を評価します。`switch` ステートメントは ActionScript ステートメントの分岐構造を作成します。

ステートメント一覧

ステートメント	説明
<code>break</code>	ループ(<code>for</code> 、 <code>for..in</code> 、 <code>do..while</code> 、または <code>while</code>)内で使用します。または、 <code>switch</code> ステートメント内の特定のケースと関連するステートメントのブロック内でも使用します。
<code>case</code>	<code>switch</code> ステートメントの条件を定義します。
<code>class</code>	カスタムクラスを定義します。自分で定義したメソッドとプロパティを共有するオブジェクトをインスタンス化できます。
<code>continue</code>	ループの終わりまで制御が通過したかのように、最も内側のループ内の残りのステートメントをすべてスキップして、ループの次の反復を開始します。
<code>default</code>	<code>switch</code> ステートメントのデフォルトケースを定義します。
<code>delete</code>	<code>reference</code> パラメータで指定されたオブジェクト参照を破棄し、参照が正常に削除された場合は <code>true</code> を返します。それ以外の場合は <code>false</code> を返します。
<code>do..while</code>	<code>while</code> ループに似ていますが、条件の最初の評価に先立ってステートメントが実行される点が異なります。
<code>dynamic</code>	指定したクラスに基づくオブジェクトが、実行時にダイナミックプロパティを追加したり、ダイナミックプロパティにアクセスできるようにします。
<code>else</code>	<code>if</code> ステートメントの条件が <code>false</code> を返したときに実行されるステートメントを指定します。
<code>else if</code>	条件を評価し、最初の <code>if</code> ステートメントの条件から <code>false</code> を返された場合に実行するステートメントを指定します。

ステートメント	説明
<code>extends</code>	他のクラス(スーパークラス)のサブクラスであるクラスを定義します。
<code>for</code>	<code>init</code> (初期化)式を1度だけ評価してから、ループシーケンスを開始します。
<code>for..in</code>	オブジェクトのプロパティまたは配列のエレメントに対して反復処理を行うもので、各プロパティまたはエレメントに対して <i>statement</i> を実行します。
<code>function</code>	何らかのタスクを実行するために定義する一連のステートメントで構成されます。
<code>get</code>	外部クラスファイルで定義したクラスに基づき、オブジェクトに関連付けられたプロパティを暗黙的に取得できるようにします。
<code>if</code>	条件を評価して、SWF ファイル内の次のアクションを決定します。
<code>implements</code>	実装するインターフェイスで宣言されているメソッドをクラスですべて定義する必要があることを指定します。
<code>import</code>	完全修飾名を指定しなくてもクラスにアクセスできるようにします。
<code>interface</code>	インターフェイスを定義します。
<code>intrinsic</code>	定義済みのクラスをコンパイル時にタイプチェックできるようにします。
<code>private</code>	変数や関数を宣言または定義しているクラス、またはそのクラスのサブクラスだからその変数や関数にアクセスできるように指定します。
<code>public</code>	すべての呼び出し元から変数や関数にアクセスできるように指定します。
<code>return</code>	関数から返される値を指定します。
<code>set</code>	外部クラスファイルで定義したクラスに基づき、オブジェクトに関連付けられたプロパティを暗黙的に設定できるようにします。
<code>set variable</code>	変数に値を代入します。
<code>static</code>	変数や関数が、1つのクラスを基に生成されるすべてのオブジェクトで生成されるのではなく、そのクラスで1回だけ生成されるようにします。
<code>super</code>	メソッドやコンストラクタのスーパークラスバージョンを呼び出します。
<code>switch</code>	ActionScript ステートメントの分岐構造を作成します。
<code>throw</code>	<code>catch()</code> コードブロックによって処理(キャッチ)できるエラーを生成(スロー)します。
<code>try..catch..finally</code>	エラーが発生する可能性のあるコードブロックを囲み、そのエラーに対処します。
<code>var</code>	ローカル変数やタイムライン変数を宣言する場合に使用します。

ステートメント	説明
<code>while</code>	条件を評価して、条件の評価結果が <code>true</code> になる場合はステートメントを実行します。その後、ループの先頭に戻り、再び条件を評価します。
<code>with</code>	<code>object</code> パラメータでムービークリップなどのオブジェクトを指定し、そのオブジェクト内の式やアクションを <code>statement(s)</code> パラメータで評価できるようになります。

break ステートメント

`break`

ループ(`for`、`for..in`、`do..while`、または`while`)内で使用します。または、`switch`ステートメント内の特定のケースと関連するステートメントのブロック内でも使用します。`break`ステートメントをループ内で使用すると、ループ本体の残りの部分をスキップし、繰り返し処理を停止して、ループステートメントの次のステートメントを実行します。`break`ステートメントを`switch`内で使用すると、`case`ブロック内の残りのステートメントをスキップし、囲んでいる`switch`ステートメントに続く最初のステートメントにジャンプします。

ネストされているループ内では、`break`ステートメントは、そのループの残りの部分をスキップするだけで、ネストされている一連のループは終了しません。一連のループを終了する方法については、`try..catch..finally`を参照してください。

例

次の例では、`break`ステートメントを使用して無限ループから抜け出します。

```
var i:Number = 0;
while (true) {
    trace(i);
    if (i >= 10) {
        break; // this will terminate/exit the loop
    }
    i++;
}
```

このコードは、次の結果を表示します。

```
0
1
2
3
4
5
6
7
8
9
10
```

関連項目

[for ステートメント](#)

case ステートメント

`case expression : statement(s)`

`switch` ステートメントの条件を定義します。`expression` パラメータと、厳密な等価(==)を使用している `switch` ステートメントの `expression` パラメータが等しい場合、`break` ステートメントが見つかるか、または `switch` ステートメントの終わりに到達するまで、`statement(s)` パラメータ内のステートメントが実行されます。

`case` ステートメントを `switch` ステートメントの外側で使用すると、エラーが発生し、スクリプトはコンパイルされません。

メモ: `statement(s)` パラメータは、必ず `break` ステートメントで終了する必要があります。

`statement(s)` パラメータの `break` ステートメントを省略すると、`switch` ステートメントが終了せずに、次の `case` ステートメントが実行されます。

パラメータ

`expression`:String - 任意の式。

例

次の例では、`switch` ステートメント `thisMonth` の条件を定義します。`thisMonth` が `case` ステートメント内の式と等しい場合は、ステートメントが実行されます。

```
var thisMonth:Number = new Date().getMonth();
switch (thisMonth) {
    case 0 :
        trace("January");
        break;
    case 1 :
        trace("February");
        break;
    case 5 :
    case 6 :
    case 7 :
        trace("Some summer month");
        break;
    case 8 :
        trace("September");
        break;
    default :
        trace("some other month");
}
```

関連項目

[break ステートメント](#)

class ステートメント

```
[dynamic] class className [ extends superClass ] [ implements interfaceName[, interfaceName... ] ] { // class definition here}
```

カスタムクラスを定義します。自分で定義したメソッドとプロパティを共有するオブジェクトをインスタンス化できます。たとえば、送り状追跡システムを作成する場合に、送り状クラスを定義して、各送り状で必要になるすべてのメソッドとプロパティを定義します。その後、`new invoice()` コマンドを使用して、送り状オブジェクトを作成します。

クラスの名前は、そのクラスが格納された外部ファイルの名前と一致している必要があります。外部ファイルの名前は、クラス名にファイル拡張子 `.as` が付けられた名前である必要があります。たとえば、あるクラスに `Student` という名前を付ける場合、このクラスを定義するファイルの名前は "`Student.as`" になります。

クラスがパッケージ化されている場合、クラス宣言には `base.sub1.sub2.MyClass` という形式で完全修飾されたクラス名を使用する必要があります。さらに、クラスの AS ファイルは "base/sub1/sub2/MyClass.as" のように、パッケージ構造を反映したディレクトリ構造のパス内に格納する必要があります。クラス定義が `class MyClass` 形式の場合、クラスはデフォルトパッケージに含まれており、"`MyClass.as`" ファイルはパス内のいずれかのディレクトリの最上位レベルに配置する必要があります。そのため、クラスを作成する前に、どのようなディレクトリ構造にするかあらかじめ決めておくことをお勧めします。そうしないと、クラスファイルを作成した後に移動する場合、新しい場所を反映するためにクラス宣言ステートメントを修正する必要が生じます。

クラス定義はネストできません。つまり、クラス定義内に別のクラスを定義することはできません。実行時にオブジェクトがダイナミックプロパティを追加したり、ダイナミックプロパティにアクセスしたりできるようにする場合は、クラスステートメントの前に `dynamic` キーワードを付加します。インターフェイスを実装するクラスを宣言するには、`implements` キーワードを使用します。クラスのサブクラスを作成するには、`extends` キーワードを使用します。クラスは1つしか拡張できませんが、インターフェイスは複数実装することができます。`implements` キーワードと `extends` キーワードは、1つのステートメントで一緒に使うことができます。次の例に、`implements` キーワードと `extends` キーワードの一般的な使い方を示します。

```
class C implements Interface_i, Interface_j // OK  
class C extends Class_d implements Interface_i, Interface_j // OK  
class C extends Class_d, Class_e // not OK
```

パラメータ

`className:String` - クラスの完全修飾名。

例

次の例では、Plant という名前のクラスを作成しています。Plant コンストラクタは 2 つのパラメータを受け取ります。

```
// Filename Plant.as
class Plant {
    // Define property names and types
    var leafType:String;
    var bloomSeason:String;
    // Following line is constructor
    // because it has the same name as the class
    function Plant(param_leafType:String, param_bloomSeason:String) {
        // Assign passed values to properties when new Plant object is created
        this.leafType = param_leafType;
        this.bloomSeason = param_bloomSeason;
    }
    // Create methods to return property values, because best practice
    // recommends against directly referencing a property of a class
    function getLeafType():String {
        return leafType;
    }
    function getBloomSeason():String {
        return bloomSeason;
    }
}
```

外部のスクリプトファイルまたは [アクション] パネルで、new 演算子を使用して Plant オブジェクトを作成します。

```
var pineTree:Plant = new Plant("Evergreen", "N/A");
// Confirm parameters were passed correctly
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());
```

次の例では、ImageLoader というクラスを作成しています。ImageLoader コンストラクタは 3 つのパラメータを受け取ります。

```
// Filename ImageLoader.as
class ImageLoader extends MovieClip {
    function ImageLoader(image:String, target_mc:MovieClip, init:Object) {
        var listenerObject:Object = new Object();
        listenerObject.onLoadInit = function(target) {
            for (var i in init) {
                target[i] = init[i];
            }
        };
        var JPEG_mcl:MovieClipLoader = new MovieClipLoader();
        JPEG_mcl.addListener(listenerObject);
        JPEG_mcl.loadClip(image, target_mc);
    }
}
```

外部のスクリプトファイルまたは [アクション] パネルで、new 演算子を使用して ImageLoader オブジェクトを作成します。

```
var jakob_mc:MovieClip = this.createEmptyMovieClip("jakob_mc",
    this.getNextHighestDepth());
var jakob:ImageLoader = new ImageLoader("http://www.helpexamples.com/flash/
    images/image1.jpg", jakob_mc, {_x:10, _y:10, _alpha:70, _rotation:-5});
```

関連項目

[dynamic ステートメント](#)

continue ステートメント

`continue`

ループの終わりまで制御が通過したかのように、最も内側のループ内の残りのステートメントをすべてスキップして、ループの次の反復を開始します。ループの外部では作用しません。

例

次の `while` ループで `continue` を使用すると、Flash インタプリタはループ本体の残りの部分をスキップし、ループの上端にジャンプします。そこで、条件が再度評価されます。

```
trace("example 1");
var i:Number = 0;
while (i < 10) {
    if (i % 3 == 0) {
        i++;
        continue;
    }
    trace(i);
    i++;
}
```

次の `do..while` ループで `continue` を使用すると、Flash インタプリタはループ本体の残りの部分をスキップし、ループの下端にジャンプします。そこで、条件が評価されます。

```
trace("example 2");
var i:Number = 0;
do {
    if (i % 3 == 0) {
        i++;
        continue;
    }
    trace(i);
    i++;
}
while (i < 10);
```

for ループで continue を使用すると、Flash インタプリタはループ本体の残りの部分をスキップします。次の例では、3 を法とする i が 0 に等しい場合、trace(i) ステートメントはスキップされます。

```
trace("example 3");
for (var i = 0; i < 10; i++) {
    if (i % 3 == 0) {
        continue;
    }
    trace(i);
}
```

次の for..in ループで continue を使用すると、Flash インタプリタはループ本体の残りの部分をスキップし、ループの上端にジャンプし、そこに列挙されている次の値が処理されます。

```
for (i in _root) {
    if (i == "$version") {
        continue;
    }
    trace(i);
}
```

関連項目

[do..while ステートメント](#)

default ステートメント

`default: statements`

switch ステートメントのデフォルトケースを定義します。このステートメントが実行されるのは、switch ステートメントの *expression* パラメータが、所定の switch ステートメントの case キーワードに続く *expression* パラメータと等しくない場合です（厳密な等価 [==] を使用）。

switch ステートメントにおいて default ケースステートメントは必須ではありません。default ケースステートメントは、リストの最後に置く必要はありません。default ステートメントを switch ステートメントの外側で使用すると、エラーが発生し、スクリプトはコンパイルされません。

パラメータ

`statements:String` - 任意のステートメント。

例

次の例では、式 A は式 B とも式 D とも等しくありません。したがって、default キーワードに続くステートメントが実行され、trace() ステートメントが [出力] パネルに送られます。

```
var dayOfWeek:Number = new Date().getDay();
switch (dayOfWeek) {
    case 1 :
        trace("Monday");
        break;
    case 2 :
        trace("Tuesday");
        break;
    case 3 :
        trace("Wednesday");
        break;
    case 4 :
        trace("Thursday");
        break;
    case 5 :
        trace("Friday");
        break;
    default :
        trace("Weekend");
}

```

関連項目

[switch ステートメント](#)

delete ステートメント

`delete reference`

reference パラメータで指定されたオブジェクト参照を破棄し、参照が正常に削除された場合は `true` を返します。それ以外の場合は `false` を返します。この演算子はスクリプトが使用しているメモリを解放するときに役立ちます。`delete` 演算子を使用してオブジェクトへの参照を削除できます。オブジェクトへの参照がすべて削除されると、オブジェクトの削除と、そのオブジェクトが使用していたメモリの解放が行われます。

`delete` は演算子ですが、通常は次のようにステートメントとして使用します。

```
delete x;
```

reference パラメータが存在しない場合、または削除できない場合には、`delete` 演算子は処理ができないので `false` を返します。定義済みオブジェクトとプロパティは削除できません。また、`var` ステートメントを使って関数内で宣言した変数も削除できません。`delete` 演算子を使用してムービークリップを削除することはできません。

戻り値

Boolean - ブール値。

パラメータ

reference:Object - 消去する変数またはオブジェクトの名前。

例

シンタックス1:次の例では、オブジェクトを作成し、使用してから、不要になったそのオブジェクトを削除します。

```
var account:Object = new Object();
account.name = "Jon";
account.balance = 10000;
trace(account.name); //output: Jon
delete account;
trace(account.name); //output: undefined
```

シンタックス2:次の例では、オブジェクトのプロパティを削除します。

```
// create the new object "account"
var account:Object = new Object();
// assign property name to the account
account.name = "Jon";
// delete the property
delete account.name;
```

シンタックス3:次の例では、オブジェクトのプロパティを削除します。

```
var my_array:Array = new Array();
my_array[0] = "abc"; // my_array.length == 1
my_array[1] = "def"; // my_array.length == 2
my_array[2] = "ghi"; // my_array.length == 3
// my_array[2] is deleted, but Array.length is not changed
delete my_array[2];
trace(my_array.length); // output: 3
trace(my_array); // output: abc,def,undefined
```

シンタックス4:次の例では、オブジェクト参照をdeleteで削除します。

```
var ref1:Object = new Object();
ref1.name = "Jody";
// copy the reference variable into a new variable
// and delete ref1
ref2 = ref1;
delete ref1;
trace("ref1.name "+ref1.name); //output: ref1.name undefined
trace("ref2.name "+ref2.name); //output: ref2.name Jody
```

`ref1` を `ref2` にコピーしていなければ、`ref1` を削除したときにオブジェクトへの参照が存在しなくなるので、オブジェクトは削除されることになります。`ref2` を削除すると、そのオブジェクトへの参照が存在しなくなるのでオブジェクトが破棄され、オブジェクトによって使用されていたメモリが使用可能になります。

関連項目

[var ステートメント](#)

do..while ステートメント

`do { statement(s) } while (condition)`

`while` ループに似ていますが、条件の最初の評価に先立ってステートメントが実行される点が異なります。その後、ステートメントは、条件が `true` と評価された場合だけ実行されます。

`do..while` ループの場合、ループ内のコードは少なくとも 1 回は必ず実行されます。`while` ループを使って、実行するステートメントのコピーを `while` ループの開始前に配置することで同じ操作を実現できますが、多くのプログラマは `do..while` ループの方が読みやすいと考えています。

条件が常に `true` と評価されると、`do..while` ループは無限ループになります。無限ループに陥ると、Flash Player に問題が発生し、警告メッセージが出力されたり、プレーヤーがクラッシュすることがあります。ループの回数が分かっている場合は、できる限り `for` ループを使用してください。`for` ループは読みやすくデバッグも簡単ですが、あらゆる状況で `do..while` に代えて使用できるわけではありません。

パラメータ

`condition:Boolean` - 評価する条件。コードの `do` ブロック内の `statement(s)` は、`condition` パラメータの評価が `true` である限り実行されます。

例

次の例では、`do..while` ループを使用して条件が `true` かどうかを評価し、`myVar` が 5 よりも大きくなるまで `myVar` をトレースします。`myVar` が 5 よりも大きくなると、ループは終了します。

```
var myVar:Number = 0;
do {
    trace(myVar);
    myVar++;
}
while (myVar < 5);
/* output:
0
1
2
3
```

```
4  
*/
```

関連項目

[break ステートメント](#)

dynamic ステートメント

```
dynamic class className [ extends superClass ] [ implements interfaceName[,  
interfaceName... ] ] { // class definition here }
```

指定したクラスに基づくオブジェクトが、実行時にダイナミックプロパティを追加したり、ダイナミックプロパティにアクセスできるようにします。

クラス定義内およびクラスインスタンス内部でアクセスされるメンバーは、クラススコープ内のメンバーと比較されないので、ダイナミッククラスに対するタイプチェックは、非ダイナミッククラスの場合よりも緩くなります。ただし、その場合でもクラスメンバー関数の戻り値とパラメータのタイプに対しては、タイプチェックが実行されます。この動作は、MovieClip オブジェクトを操作する場合に特に便利です。ムービークリップにプロパティやオブジェクトを動的に追加する方法が数多くあります。具体的には、MovieClip.createEmptyMovieClip() や MovieClip.createTextField() などです。

ダイナミッククラスのサブクラスもダイナミックになります。

例

次の例では、クラス Person2 が dynamic キーワードで宣言されていないので、このクラスで宣言されていない関数を呼び出すと、コンパイルエラーが発生します。

```
class Person2 {  
    var name:String;  
    var age:Number;  
    function Person2(param_name:String, param_age:Number) {  
        trace ("anything");  
        this.name = param_name;  
        this.age = param_age;  
    }  
}
```

同じディレクトリ内にある FLA または AS ファイル内で、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
// Before dynamic is added  
var craig:Person2 = new Person2("Craiggers", 32);  
for (i in craig) {  
    trace("craig." + i + " = " + craig[i]);  
}  
/* output:
```

```
craig.age = 32
craig.name = Craiggers */

宣言されていない関数 dance を追加すると、次に示すように、エラーが生成されます。

trace("");
craig.dance = true;
for (i in craig) {
    trace("craig." + i + " = " + craig[i]);
}
/* output: **Error** Scene=Scene 1, layer=Layer 1, frame=1:Line 14: There is no
   property with the name 'dance'. craig.dance = true; Total ActionScript Errors:
   1 Reported Errors: 1 */

dynamic キーワードを Person2 クラスに追加します。最初の行は次のようにになります。

dynamic class Person2 {

コードを再びテストします。出力は次のようにになります。

craig.dance = true craig.age = 32 craig.name = Craiggers
```

関連項目

[class ステートメント](#)

else ステートメント

```
if (condition){ statement(s); } else { statement(s); }
```

if ステートメントの条件が `false` を返したときに実行されるステートメントを指定します。else ステートメントによって実行するステートメントブロックを囲む中括弧(`{}`)は、実行するステートメントが1つしかない場合は必要ありません。

パラメータ

`condition: Boolean` - 評価結果が `true` または `false` になる式。

例

次の例では、`else` 条件を使用して、`age_txt` 変数が 18 よりも大きいかまたは小さいかを判定しています。

```
if (age_txt.text>=18) { trace("welcome, user"); } else { trace("sorry, junior");
    userObject.minor = true; userObject.accessAllowed = false; }
```

次の例では、`else` ステートメントに続くステートメントが1つだけなので、中括弧(`{}`)は必要ありません。

```
if (age_txt.text>18) { trace("welcome, user"); } else trace("sorry, junior");
```

関連項目

[if ステートメント](#)

else if ステートメント

```
if (condition){ statement(s); }
else if (condition){ statement(s); }
```

条件を評価し、最初の if ステートメントの条件から false を返された場合に実行するステートメントを指定します。else if 条件から true が返されると、Flash インタプリタは中括弧({ })内の条件に続くステートメントを実行します。else if 条件が false の場合は、中括弧内のステートメントはスキップされ、中括弧の後のステートメントが実行されます。

スクリプト内に分岐処理を作成するには elseif ステートメントを使用します。分岐が複数ある場合は、switch ステートメントの使用を検討してください。

パラメータ

condition:Boolean - 評価結果が true または false になる式。

例

次の例では、else if ステートメントを使用して、score_txt を指定された値と比較しています。

```
if (score_txt.text>90) { trace("A"); } else if (score_txt.text>75) { trace("B"); }
} else if (score_txt.text>60) { trace("C"); } else { trace("F"); }
```

関連項目

[if ステートメント](#)

extends ステートメント

シンタックス1:

```
class className extends otherClassName {}
```

シンタックス2:

```
interface interfaceName extends otherInterfaceName {}
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

他のクラス（スーパークラス）のサブクラスであるクラスを定義します。サブクラスは、スーパークラスで定義されているメソッド、プロパティ、関数などをすべて継承します。

インターフェイスは extends キーワードを使用して拡張することもできます。他のインターフェイスを拡張したインターフェイスには、元のインターフェイスのすべてのメソッド宣言がすべて含まれます。

パラメータ

className:String - 定義するクラスの名前。

例

次の例の Car クラスは Vehicle クラスを拡張しているので、Vehicle クラスのすべてのメソッド、プロパティ、および関数を継承します。スクリプトで Car オブジェクトをインスタンス化すると、Car クラスのメソッドと Vehicle クラスのメソッドの両方を使用できます。

次の例では、Vehicle クラスを定義している "Vehicle.as" ファイルの内容を示します。

```
class Vehicle {  
    var numDoors:Number;  
    var color:String;  
    function Vehicle(param_numDoors:Number, param_color:String) {  
        this.numDoors = param_numDoors;  
        this.color = param_color;  
    }  
    function start():Void {  
        trace("[Vehicle] start");  
    }  
    function stop():Void {  
        trace("[Vehicle] stop");  
    }  
    function reverse():Void {  
        trace("[Vehicle] reverse");  
    }  
}
```

次の例では、同じディレクトリ内にあるもう1つの AS ファイル "Car.as" の内容を示します。このクラスは Vehicle クラスを拡張したクラスで、次のような3つの変更を行っています。Car クラスでは、最初に、車オブジェクトにフルサイズのスペアタイヤが装着されているかどうかを追跡するための変数 fullSizeSpare を追加しています。2番目に、車の盗難防止アラームをアクティブにする activateCarAlarm() を車に固有の新しいメソッドとして追加しています。3番目に、Car クラスでは停車の際にアンチロックブレーキシステムを使用することを示すために、stop() 関数が上書きされています。

```
class Car extends Vehicle {  
    var fullSizeSpare:Boolean;  
    function Car(param_numDoors:Number, param_color:String,  
        param_fullSizeSpare:Boolean) {  
        this.numDoors = param_numDoors;  
        this.color = param_color;  
        this.fullSizeSpare = param_fullSizeSpare;  
    }  
    function activateCarAlarm():Void {  
        trace("[Car] activateCarAlarm");  
    }  
    function stop():Void {
```

```
    trace("[Car] stop with anti-lock brakes");
}
}
```

次の例では、Car オブジェクトをインスタンス化し、Vehicle クラスに定義されているメソッド (start()) を呼び出します。次に、Car クラスで上書きされたメソッド (stop()) を呼び出し、最後に、Car クラスのメソッド (activateCarAlarm()) を呼び出します。

```
var myNewCar:Car = new Car(2, "Red", true);
myNewCar.start(); // output: [Vehicle] start
myNewCar.stop(); // output: [Car] stop with anti-lock brakes
myNewCar.activateCarAlarm(); // output: [Car] activateCarAlarm
```

Vehicle クラスのサブクラスは、super キーワードを使用して記述することもできます。サブクラスは、このキーワードを使用することでスーパークラスのプロパティやメソッドにアクセスできます。次の例では、同じディレクトリ内にある 3 番目の AS ファイル "Truck.as" の内容を示します。Truck クラスでは、コンストラクタ内と、上書きされた reverse() 関数内で super キーワードを使用しています。

```
class Truck extends Vehicle {
    var numWheels:Number;
    function Truck(param_numDoors:Number, param_color:String,
        param_numWheels:Number) {
        super(param_numDoors, param_color);
        this.numWheels = param_numWheels;
    }
    function reverse():Void {
        beep();
        super.reverse();
    }
    function beep():Void {
        trace("[Truck] make beeping sound");
    }
}
```

次の例では、Truck オブジェクトをインスタンス化し、Truck クラスで上書きされたメソッド (reverse()) を呼び出してから、Vehicle クラスで定義されているメソッド (stop()) を呼び出しています。

```
var myTruck:Truck = new Truck(2, "White", 18);
myTruck.reverse(); // output: [Truck] make beeping sound [Vehicle] reverse
myTruck.stop(); // output: [Vehicle] stop
```

関連項目

[class ステートメント](#)

for ステートメント

```
for(init; condition; next) {  
  statement(s);  
}
```

init(初期化)式を1度だけ評価してから、ループシーケンスを開始します。ループシーケンスは、*condition*式を評価することで開始されます。*condition*式の評価結果がtrueの場合は、*statement*が実行され、*next*式が評価されます。その後、*condition*式の評価からループシーケンスが再び開始されます。

forステートメントによって実行するステートメントブロックを囲む中括弧(())は、実行するステートメントが1つしかない場合は必要ありません。

パラメータ

init- ループの開始前に評価される式。通常は代入式です。このパラメータに対しては、varステートメントも実行できます。

例

次の例では、forを使用して配列のエレメントを追加します。

```
var my_array:Array = new Array();  
for (var i:Number = 0; i < 10; i++) {  
  my_array[i] = (i + 5) * 10;  
}  
trace(my_array); // output: 50,60,70,80,90,100,110,120,130,140
```

次の例では、forを使って同じアクションを繰り返し実行します。次のコードでは、forループにより1から100の数値を加算します。

```
var sum:Number = 0;  
for (var i:Number = 1; i <= 100; i++) {  
  sum += i;  
}  
trace(sum); // output: 5050
```

次の例では、実行するステートメントが1つしかない場合は中括弧を付ける必要がないことを示します。

```
var sum:Number = 0;  
for (var i:Number = 1; i <= 100; i++)  
  sum += i;  
trace(sum); // output: 5050
```

関連項目

[++ インクリメント演算子](#)

for..in ステートメント

```
for (variableIterant in object) {  
    statement(s);  
}
```

オブジェクトのプロパティまたは配列のエレメントに対して反復処理を行うもので、各プロパティまたはエレメントに対して *statement* を実行します。for..in アクションでは、オブジェクトのメソッドは列挙されません。

プロパティの中には、for..in アクションで列挙できないものがあります。たとえば、_x や _y のようなムービークリッププロパティは列挙されません。外部クラスファイルでは、インスタンスマンバーとは異なり、静的メンバーは列挙されません。

for..in ステートメントは、反復処理されるオブジェクトのプロトタイプチェーン内のオブジェクトのプロパティに対して反復します。オブジェクトのプロパティが最初に列挙され、次にそのプロトタイプのプロパティが列挙され、さらにそのプロトタイプのプロトタイプのプロパティが列挙されます。以下同様です。for..in ステートメントは、同じプロパティ名を繰り返して列挙しません。オブジェクト child にプロトタイプ parent が存在し、両方に prop プロパティがある場合、child の for..in ステートメントは、child の prop プロパティは列挙しますが parent の prop プロパティは無視します。for..in ステートメントによって実行するステートメントブロックを囲む中括弧 (()) は、実行するステートメントが1つしかない場合は必要ありません。

for..in ループをクラスファイル(外部 AS ファイル)内に記述した場合、インスタンスマンバーをループで使用することはできませんが、静的メンバーはループで使用できます。ただし、クラスのインスタンスの FLA ファイル内に for..in ループを記述した場合、インスタンスマンバーは使用できますが、静的メンバーは使用できません。

パラメータ

variableIterant:String - 反復子の役割を果たし、オブジェクトのプロパティまたは配列内のエレメントを参照する変数の名前。

例

次の例では、for..in を使用したオブジェクトのプロパティに対する繰り返し処理を示します。

```
var myObject:Object = {firstName:"Tara", age:27, city:"San Francisco"};  
for (var prop in myObject) {  
    trace("myObject."+prop+" = "+myObject[prop]);  
}  
//output  
myObject.firstName = Tara  
myObject.age = 27  
myObject.city = San Francisco
```

次の例では、`for..in`を使用した配列のエレメントに対する繰り返し処理を示します。

```
var myArray:Array = new Array("one", "two", "three");
for (var index in myArray)
    trace("myArray["+index+"] = " + myArray[index]);
// output:
myArray[2] = three
myArray[1] = two
myArray[0] = one
```

次の例では、`for..in`で`typeof`演算子を使用して特定のタイプの子に対して反復処理を行います。

```
for (var name in this) {
    if (typeof (this[name]) == "movieclip") {
        trace("I have a movie clip child named "+name);
    }
}
```

メモ: ムービークリップが複数ある場合、その出力はそれらのクリップのインスタンス名で構成されます。

次の例では、ムービークリップの子を列挙し、それを対応するタイムラインのフレーム 2 に進めます。`RadioButtonGroup` ムービークリップは親であり、`_RedRadioButton_`、`_GreenRadioButton_`、および`_BlueRadioButton_` という複数の子を持っています。

```
for (var name in RadioButtonGroup) { RadioButtonGroup[name].gotoAndStop(2); }
```

function ステートメント

シンタックス1:(指定した関数を宣言します)
`function functionname([parameter0, parameter1,...parameterN]) {statement(s)}` シンタックス2:(匿名関数を宣言し、その関数への参照を返します)
`function ([parameter0, parameter1,...parameterN]) {statement(s)}`

何らかのタスクを実行するために定義する一連のステートメントで構成されます。ある位置で関数を定義し、SWF ファイルの異なるスクリプトからその関数を呼び出すことができます。関数を定義する場合、その関数のパラメータも指定できます。パラメータは、関数が処理する値のプレースホルダです。関数を呼び出すたびに異なるパラメータを渡すことができます。これにより、1つの関数を多くの異なる状況で再利用できます。

関数で値を生成する、つまり "返す" には、関数の`statement(s)`で`return`ステートメントを使用します。

このステートメントを使用して、`functionname`、`parameters`、および`statement(s)`が指定された`function`を定義します。スクリプトによって関数が呼び出されると、関数定義内のステートメントが実行されます。関数は前方参照が許されます。つまり、同じスクリプト内では、関数を呼び出す箇所よりも後に関数を宣言できます。関数定義は、同じ関数の以前の定義と置き換わります。ステートメントが許されている場所であればどこでも、このシンタックスを使用できます。

このステートメントを使用して匿名関数を作成し、その関数への参照を返すこともできます。このシンタックスは式の中で使用され、特にオブジェクト内にメソッドを組み込む場合に便利です。

さらに、関数定義内で arguments オブジェクトを使用することができます。一般に、arguments オブジェクトは、可変数のパラメータを受け取る関数や再帰的な匿名関数を作成する場合に使用します。

戻り値

String - シンタックス 1: この宣言形式では何も返されません。シンタックス 2: 匿名関数への参照。

パラメータ

functionname:String - 宣言した関数の名前。

例

次の例では、1つのパラメータを受け取る関数 sqr を定義し、そのパラメータの Math.pow(x, 2) を返します。

```
function sqr(x:Number) {  
    return Math.pow(x, 2);  
}  
var y:Number = sqr(3);  
trace(y); // output: 9
```

同じスクリプト内で関数を定義および使用する場合は、関数の使用箇所の後に関数定義を記述することもできます。

```
var y:Number = sqr(3);  
trace(y); // output: 9  
function sqr(x:Number) {  
    return Math.pow(x, 2);  
}
```

次の関数は、LoadVars オブジェクトを作成し、"params.txt" を SWF ファイルにロードします。ファイルが正常にロードされると、"variables loaded" と表示されます。

```
var myLV:LoadVars = new LoadVars();  
myLV.load("params.txt");  
myLV.onLoad = function(success:Boolean) {  
    trace("variables loaded");  
}
```

get ステートメント

```
function get property () { // your statements here }
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

外部クラスファイルで定義したクラスに基づき、オブジェクトに関連付けられたプロパティを暗黙的に取得できるようにします。暗黙的な get メソッドを使用すると、プロパティに直接アクセスすることなく、オブジェクトのプロパティにアクセスできます。暗黙的な get メソッドと set メソッドは、ActionScript 1 の Object.addProperty() メソッドの簡易版です。

パラメータ

property:String - get でアクセスするプロパティの参照名。この値は対応する set コマンドに使用した値と一致させる必要があります。

例

次の例では、Team クラスを定義します。Team クラスには、クラス内のプロパティを取得および設定するための get/set メソッドがあります。

```
class Team {
    var teamName:String;
    var teamCode:String;
    var teamPlayers:Array = new Array();
    function Team(param_name:String, param_code:String) {
        this.teamName = param_name;
        this.teamCode = param_code;
    }
    function get name():String {
        return this.teamName;
    }
    function set name(param_name:String):Void {
        this.teamName = param_name;
    }
}
```

次の ActionScript をタイムラインのフレームに入力します。

```
var giants:Team = new Team("San Fran", "SFO");
trace(giants.name);
giants.name = "San Francisco";
trace(giants.name);
/* output:
San Fran San Francisco */
giants.name をトレースするときは、get メソッドを使用してプロパティの値を返します。
```

関連項目

[addProperty \(Object.addProperty メソッド\)](#)

if ステートメント

```
if(condition) { statement(s); }
```

条件を評価して、SWF ファイル内の次のアクションを決定します。条件が `true` の場合は、条件に続く中括弧(`{}`)内のステートメントが実行されます。条件が `false` の場合は、中括弧内のステートメントはスキップされ、中括弧の後のステートメントが実行されます。スクリプト内に分岐処理を作成するには、`if` ステートメントを `else` ステートメントや `else if` ステートメントと組み合わせます。
`if` ステートメントによって実行するステートメントブロックを囲む中括弧(`{}`)は、実行するステートメントが1つしかない場合は必要ありません。

パラメータ

`condition: Boolean` - 評価結果が `true` または `false` になる式。

例

次の例では、括弧内の条件で変数 `name` を評価し、リテラル値 "Erica" が含まれているかどうかを確認します。"Erica" が含まれている場合は、中括弧内の `play()` 関数が実行されます。

```
if(name == "Erica"){
    play();
}
```

次の例では、`if` ステートメントを使用して、ユーザーが SWF ファイル内の `submit_btn` インスタンスをクリックするまでに要した時間を評価します。SWF ファイルの再生開始後 10 秒を超えてからユーザーがボタンをクリックした場合、条件は `true` と評価され、(`createTextField()` を使用して) 実行時に作成されたテキストフィールドに、中括弧(`{}`)内のメッセージが表示されます。SWF ファイルの再生開始後 10 秒以内にユーザーがボタンをクリックした場合、条件は `false` と評価され、別のメッセージが表示されます。

```
this.createTextField("message_txt", this.getNextHighestDepth, 0, 0, 100, 22);
message_txt.autoSize = true;
var startTime:Number = getTimer();
this.submit_btn.onRelease = function() {
    var difference:Number = (getTimer() - startTime) / 1000;
    if (difference > 10) {
        this._parent.message_txt.text = "Not very speedy, you took "+difference+
            " seconds.";
    }
    else {
        this._parent.message_txt.text = "Very good, you hit the button in "+difference+
            " seconds.";
    }
};
```

関連項目

[else ステートメント](#)

implements ステートメント

`myClass implements interface01 [, interface02 , ...]`

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

実装するインターフェイスで宣言されているメソッドをクラスですべて定義する必要があることを指定します。

例

詳細については、[interface](#) を参照してください。

関連項目

[class ステートメント](#)

import ステートメント

`import className import packageName.*`

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このステートメントは、[アクション] パネルおよび外部クラスファイルで使用できます。

完全修飾名を指定しなくてもクラスにアクセスできるようにします。たとえば、スクリプト内でカスタムクラス macr.util.users.UserClass を使用する場合、このクラスを完全修飾名で参照するか、`import` で読み込む必要があります。`import` で読み込んでおくと、クラス名での参照が可能になります。

```
// before importing
var myUser:macr.util.users.UserClass = new macr.util.users.UserClass();
// after importing
import macr.util.users.UserClass;
var myUser:UserClass = new UserClass();
```

アクセス対象のクラスファイルがパッケージ (`working_directory/macr/utils/users`) 内に複数存在する場合は、次の例に示すように、1つのステートメントですべてのクラスファイルを読み込むことができます。

```
import macr.util.users.*;
import ステートメントは最初に指定しておく必要があります。そうすると、このステートメントで読み込まれたクラスは以降、完全修飾名を使わずにアクセスできます。
```

読み込んだクラスがスクリプト内で使用されなかった場合、そのクラスは SWF ファイルには出力されません。したがって、SWF ファイルのサイズを気にすることなく、大きなパッケージを読み込むことができます。クラスに関連付けられたバイトコードは、それが実際に使用された場合にのみ、SWF ファイルに含まれます。

import ステートメントは、それを呼び出している現在のスクリプト(フレームまたはオブジェクト)にのみ適用されます。たとえば、macr.util パッケージのすべてのクラスを Flash ドキュメントのフレーム 1 に読み込んだと仮定します。そのフレームでは、そのパッケージ内のクラスを簡単な名前で参照できます。

```
// On Frame 1 of a FLA:  
import macr.util.*;  
var myFoo:foo = new foo();
```

ただし、他のフレームのスクリプトでは、そのパッケージのクラスを完全修飾名(var myFoo:foo = new macr.util.foo();)で参照するか、そのパッケージのクラスを読み込む import ステートメントを追加する必要があります。

パラメータ

className:String - 外部クラスファイルで定義したクラスの完全修飾名。

例

interface ステートメント

```
interface InterfaceName [extends InterfaceName ] {}
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

インターフェイスを定義します。インターフェイスはクラスに似ていますが、次に示す重要な違いがあります。

- インターフェイスにはメソッドの宣言だけが含まれます。メソッド実装は含まれません。つまり、インターフェイスを実装するすべてのクラスは、インターフェイスで宣言されている各メソッドの実装を定義する必要があります。
- インターフェイス定義では、パブリックメンバーしか使用できず、インスタンスおよびクラスのメンバーは使用できません。
- get ステートメントおよび set ステートメントは、インターフェイス定義では使用できません。

例

次の例では、インターフェイスを定義および実装する方法をいくつか示しています。

```
(in top-level package .as files Ia, B, C, Ib, D, Ic, E)
// filename Ia.as
interface Ia {
    function k():Number; // method declaration only
    function n(x:Number):Number; // without implementation
}
// filename B.as
class B implements Ia {
    function k():Number {
        return 25;
    }
    function n(x:Number):Number {
        return x + 5;
    }
} // external script or Actions panel // script file
var mvar:B = new B();
trace(mvar.k()); // 25
trace(mvar.n(7)); // 12
// filename C.as
class C implements Ia {
    function k():Number {
        return 25;
    }
} // error: class must implement all interface methods
// filename Ib.as
interface Ib {
    function o():Void;
}
class D implements Ia, Ib {
    function k():Number {
        return 15;
    }
    function n(x:Number):Number {
        return x * x;
    }
    function o():Void {
        trace("o");
    }
} // external script or Actions panel // script file
mvar = new D();
trace(mvar.k()); // 15
trace(mvar.n(7)); // 49
trace(mvar.o()); // "o"
interface Ic extends Ia {
    function p():Void;
}
class E implements Ib, Ic {
```

```
function k():Number {
    return 25;
}
function n(x:Number):Number {
    return x + 5;
}
function o():Void {
    trace("o");
}
function p():Void {
    trace("p");
}
```

関連項目

[class ステートメント](#)

intrinsic ステートメント

```
intrinsic class className [extends superClass] [implements interfaceName [,  
    interfaceName...] ] {  
    //class definition here  
}
```

定義済みのクラスをコンパイル時にタイプチェックできるようにします。Flash では、intrinsic クラス宣言を使用することで、Array、Object、および String といったビルトインクラスのタイプチェックをコンパイル時に行えます。このキーワードは、関数の実装が必要でないこと、およびバイトコードを生成しないことをコンパイラに示します。

intrinsic キーワードは、変数や関数の宣言に使用することもできます。Flash では、このキーワードを使用することで、グローバル関数やグローバルプロパティのタイプチェックをコンパイル時に行えます。

intrinsic キーワードは、ビルトインクラス、ビルトインオブジェクト、グローバル変数、グローバル関数のタイプチェックをコンパイル時に行えるようにする目的で作成されたものです。このキーワードは汎用的に使用するというものではなく、定義済みのクラス（特に ActionScript 1.0 を使用して定義されているクラス）をコンパイル時にタイプチェックする方法を模索している開発者にとって有用なものです。

このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

例

次の例では、定義済みの ActionScript 1.0 クラスのコンパイル時タイプチェックを有効にしています。このコードでは、`myCircle.setRadius()` 呼び出しにより Number 値ではなく String 値がパラメータとして送られるので、コンパイル時エラーが発生します。パラメータを Number 値に（たとえば "10" を 10 に）変換することで、このエラーを回避できます。

```
// The following code must be placed in a file named Circle.as
// that resides within your classpath:
intrinsic class Circle {
    var radius:Number;
    function Circle(radius:Number);
    function getArea():Number;
    function getDiameter():Number;
    function setRadius(param_radius:Number):Number;
}

// This ActionScript 1.0 class definition may be placed in your FLA file.
// Circle class is defined using ActionScript 1.0
function Circle(radius) {
    this.radius = radius;
    this.getArea = function(){
        return Math.PI*this.radius*this.radius;
    };
    this.getDiameter = function() {
        return 2*this.radius;
    };
    this.setRadius = function(param_radius) {
        this.radius = param_radius;
    }
}

// ActionScript 2.0 code that uses the Circle class
var myCircle:Circle = new Circle(5);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
myCircle.setRadius("10");
trace(myCircle.radius);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
```

関連項目

[class ステートメント](#)

private ステートメント

```
class someClassName{  
    private var name;  
    private function name() {  
        // your statements here  
    }  
}
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

変数や関数を宣言または定義しているクラス、またはそのクラスのサブクラスだからその変数や関数にアクセスできるように指定します。デフォルトでは、すべての呼び出し元から変数や関数にアクセスできます。このキーワードは、変数や関数へのアクセスを制限する場合に使用します。

このキーワードは、クラス定義でのみ使用できます。インターフェイス定義では使用できません。

パラメータ

name:String - private として指定する変数または関数の名前。

例

次の例では、private キーワードを使用して、クラス内の特定のプロパティを非表示にする方法を示します。"Login.as" という名前で新しい AS ファイルを作成します。

```
class Login {  
    private var loginUserName:String;  
    private var loginPassword:String;  
    public function Login(param_username:String, param_password:String) {  
        this.loginUserName = param_username;  
        this.loginPassword = param_password;  
    }  
    public function get username():String {  
        return this.loginUserName;  
    }  
    public function set username(param_username:String):Void {  
        this.loginUserName = param_username;  
    }  
    public function set password(param_password:String):Void {  
        this.loginPassword = param_password;  
    }  
}
```

"Login.as" と同じディレクトリに、新しい FLA ドキュメントまたは AS ドキュメントを作成します。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
import Login;
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
trace(gus.password); // output: undefined
trace(gus.loginPassword); // error
```

loginPassword はプライベート変数なので、"Login.as" クラスファイルの外側からこの変数にアクセスすることはできません。プライベート変数にアクセスしようとすると、エラーメッセージが表示されます。

関連項目

[public ステートメント](#)

public ステートメント

```
class someClassName{ public var name; public function name() { // your statements
    here } }
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

すべての呼び出し元から変数や関数にアクセスできるように指定します。デフォルトでは変数および関数が `public` として扱われるため、このキーワードは主に形式上の目的で使用します。たとえば、`private` または `static` の変数が含まれるコードブロックで `public` を明確に区別したい場合などが該当します。

パラメータ

`name:String - public` として指定する変数または関数の名前。

例

次の例では、クラスファイルでパブリック変数を使用する方法を示します。`User.as` という名前で新しいクラスファイルを作成し、次のコードを入力します。

```
class User {
    public var age:Number;
    public var name:String;
}
```

次に、同じディレクトリに新しい FLA または AS ファイルを作成し、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
import User;
var jimmy:User = new User();
jimmy.age = 27;
jimmy.name = "jimmy";
```

User クラスのパブリック変数の 1 つをプライベート変数に変更した場合は、プロパティにアクセスしようとするとエラーが生成されます。

関連項目

[private ステートメント](#)

return ステートメント

`return[expression]`

関数から返される値を指定します。return ステートメントは、*expression* を評価し、その結果をステートメントが実行される関数の値として返します。return ステートメントにより、実行は即座に呼び出し元の関数に返されます。return ステートメントを単独で使用したときの戻り値は `undefined` です。

複数の値を取得することはできません。複数の値を取得しようとした場合は、最後の値だけが返されます。たとえば、次の例では `c` が返されます。

```
return a, b, c;
```

複数の値を取得する必要がある場合は、代わりに配列またはオブジェクトを使用します。

戻り値

`String - expression` パラメータ (指定されている場合) の評価。

パラメータ

`expression` - 関数の値として評価して返すストリング、数値、ブール値、配列、またはオブジェクト。このパラメータはオプションです。

例

次の例では、`sum()` 関数の本体内で return ステートメントを使用し、3 つのパラメータの加算結果を返します。コードの 2 行目で `sum()` 関数を呼び出し、戻り値を変数 `newValue` に代入します。

```
function sum(a:Number, b:Number, c:Number):Number {
    return (a + b + c);
}
var newValue:Number = sum(4, 32, 78);
trace(newValue); // output: 114
```

関連項目

[function ステートメント](#)

set ステートメント

```
function set property(varName) {  
    // your statements here  
}
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

外部クラスファイルで定義したクラスに基づき、オブジェクトに関連付けられたプロパティを暗黙的に設定できるようにします。set メソッドを暗黙的に使用することにより、オブジェクトのプロパティに直接アクセスせずにプロパティの値を変更できます。暗黙的な get メソッドと set メソッドは、ActionScript 1 の Object.addProperty() メソッドの簡易版です。

パラメータ

property:String - set でアクセスするプロパティの参照名。この値は対応する get コマンドで使用する値と一致させる必要があります。

例

次の例では、Login クラスを作成し、set キーワードを使用してプライベート変数を設定する方法を示します。

```
class Login {  
    private var loginUserName:String;  
    private var loginPassword:String;  
    public function Login(param_username:String, param_password:String) {  
        this.loginUserName = param_username;  
        this.loginPassword = param_password;  
    }  
    public function get username():String {  
        return this.loginUserName;  
    }  
    public function set username(param_username:String):Void {  
        this.loginUserName = param_username;  
    }  
    public function set password(param_password:String):Void {  
        this.loginPassword = param_password;  
    }  
}
```

"Login.as" と同じディレクトリにある FLA ファイルまたは AS ファイルで、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
gus.username = "Rupert";
trace(gus.username); // output: Rupert
```

この例では、値がトレースされると get 関数が実行されます。set 関数は、次のように値が渡される場合にのみトリガれます。

```
gus.username = "Rupert";
```

関連項目

[get ステートメント](#)

set variable ステートメント

```
set("variableString", expression)
```

変数に値を代入します。変数とは、情報を格納する容器のようなものです。容器そのものは変化しませんが、その内容は変化します。SWF ファイルの再生時に変数の値を変更すると、ユーザーが実行した処理に関する情報を記録および保存できます。また、SWF ファイルの再生時に変化した値を記録し、条件が true(真)なのか false(偽)なのかを評価することもできます。

変数には、ストリング、数値、ブール値、オブジェクト、ムービークリップなどのすべてのデータ型を保持できます。各 SWF ファイルおよびムービークリップのタイムラインには、それぞれ独自の変数のセットがあり、各変数の値は他のタイムラインの変数には影響されません。

set ステートメント内では厳密な型指定はサポートされません。このステートメントを使用して、クラスファイル内の変数に関連付けられたデータ型と異なるデータ型を持つ値を変数に設定した場合、コンパイルエラーは生成されません。

variableString パラメータは、変数名ではなくストリングです。この違いは重要なので注意が必要です。最初のパラメータとして既存の変数名を set() に渡すときに引用符 ("") で囲まないと、変数の評価結果が *expression* の値として代入されます。たとえば、*myVariable* という名前のストリング変数を作成し、"Tuesday" という値を代入するとします。このときに不注意で引用符を使用しないと、新しい変数 *Tuesday* が作成され、*myVariable* に代入しようとした値がその変数に代入されることになります。

```
var myVariable:String = "Tuesday";
set (myVariable, "Saturday");
trace(myVariable); // outputs Tuesday
trace(Tuesday); // outputs Saturday
```

引用符 ("") を使用すれば、この問題を回避できます。

```
set ("myVariable", "Saturday");
trace(myVariable); //outputs Saturday
```

パラメータ

`variableString:String - expression` パラメータの値を保持する変数を指定するストリング。

例

次の例では、変数に値を代入します。"Jakob" という値を name 変数に割り当てます。

```
set("name", "Jakob");
trace(name);
```

次のコードは 3 回ループして、3 つの変数(caption0、caption1、caption2)を作成します。

```
for (var i = 0; i < 3; i++) {
    set("caption" + i, "this is caption " + i);
}
trace(caption0);
trace(caption1);
trace(caption2);
```

関連項目

[var ステートメント](#)

static ステートメント

```
class someClassName{ static var name; static function name() { // your statements
here } }
```

メモ: このキーワードを使用するには、FLA ファイルの [パブリッシュ設定] ダイアログボックスの [Flash] タブで、ActionScript 2.0 および Flash Player 6 以降を指定する必要があります。このキーワードは、[アクション] パネルに記述するスクリプトではありません。外部スクリプトファイルで使用する場合にのみサポートされます。

変数や関数が、1 つのクラスを基に生成されるすべてのオブジェクトで生成されるのではなく、そのクラスで 1 回だけ生成されるようにします。

シンタックス `someClassName.name` を使用すると、クラスのインスタンスを作成せずに静的クラスメンバーにアクセスできます。クラスのインスタンスを作成する場合は、そのインスタンスを使用して静的メンバーにアクセスすることもできます。ただし、その静的メンバーにアクセスする非静的関数を使用する場合のみに限定されます。

このキーワードは、クラス定義でのみ使用できます。インターフェイス定義では使用できません。

パラメータ

`name:String - static` として指定する変数または関数の名前。

例

次の例では、作成されたクラスのインスタンス数を追跡するカウンタを、`static` キーワードを使用して作成する方法を示します。`numInstances` 変数は静的なので、個々のインスタンスごとに作成されるのではなく、クラス全体で1回のみ作成されます。`"Users.as"` という新しい AS ファイルを作成し、次のコードを入力します。

```
class Users {  
    private static var numInstances:Number = 0;  
    function Users() {  
        numInstances++;  
    }  
    static function get instances():Number {  
        return numInstances;  
    }  
}
```

同じディレクトリに FLA ドキュメントまたは AS ドキュメントを作成し、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
trace(Users.instances);  
var user1:Users = new Users();  
trace(Users.instances);  
var user2:Users = new Users();  
trace(Users.instances);
```

関連項目

[private ステートメント](#)

super ステートメント

```
super.method([arg1, ..., argN])  
super([arg1, ..., argN])
```

最初のシンタックススタイルは、オブジェクトメソッドの本体内で使用して、メソッドのスーパークラスバージョンを呼び出すことができます。必要に応じて、スーパークラスメソッドにパラメータ (`arg1 ... argN`) を渡すこともできます。このスタイルは、スーパークラスのメソッドにビヘイビアを追加するだけでなく、そのメソッドを使用して元のビヘイビアを実行するようなサブクラスメソッドを作成する場合に便利です。

2番目のシンタックススタイルは、コンストラクタ関数の本体内で使用して、コンストラクタ関数のスーパークラスバージョンを呼び出すことができます。必要に応じて、関数にパラメータを渡すこともできます。このスタイルは、補足的な初期化を行うだけでなく、スーパークラスのコンストラクタを呼び出してスーパークラスの初期化を行うようなサブクラスを作成する場合に便利です。

戻り値

どちらの形式も関数を呼び出します。関数は不特定の値を返します。

パラメータ

`method:Function` - スーパークラスで呼び出すメソッド。

`argN` - メソッドのスーパークラスバージョン(シンタックス1)またはスーパークラスのコンストラクタ関数(シンタックス2)に渡すオプションのパラメータ。

switch ステートメント

```
switch (expression){caseClause: [defaultClause:] }
```

ActionScript ステートメントの分岐構造を作成します。if ステートメントと同様に、switch ステートメントは条件をテストし、条件が `true` を返した場合にステートメントを実行します。すべての switch ステートメントには、`default` ケースを入れるようにしてください。`default` ケースには `break` ステートメントを入れて、後で別の `case` を追加した場合にフォールスルーエラーが発生しないようにしてください。`case` がフォールスルーエラーを起こすのは、`break` ステートメントが含まれていない場合です。

パラメータ

`expression` - 任意の式。

例

次の例では、`String.fromCharCode(Key.getAscii())` パラメータが A と評価された場合、`case "A"` の後ろにある `trace()` ステートメントが実行されます。このパラメータが a と評価された場合、`case "a"` の後ろにある `trace()` ステートメントが実行されます。以降、同様に処理されます。いずれの `case` 式も `String.fromCharCode(Key.getAscii())` パラメータと一致しない場合は、`default` キーワードの後にある `trace()` ステートメントが実行されます。

```
var listenerObj:Object = new Object();
listenerObj.onKeyDown = function() {
    switch (String.fromCharCode(Key.getAscii())) {
        case "A" :
            trace("you pressed A");
            break;
        case "a" :
            trace("you pressed a");
            break;
        case "E" :
        case "e" :
            trace("you pressed E or e");
            break;
        case "I" :
        case "i" :
            trace("you pressed I or i");
            break;
        default :
    }
}
```

```
trace("you pressed some other key");
break;
}
};

Key.addListener(listenerObj);
```

関連項目

[== 厳密な等価演算子](#)

throw ステートメント

throw expression

`catch{}` コードブロックによって処理(キャッチ)できるエラーを生成(スロー)します。`catch` ブロックが例外をキャッチしない場合は、スローされた値のストリング表現が[出力]パネルに表示されます。

一般には、`Error` クラスまたはそのサブクラスのインスタンスをスローします。「例」を参照してください。

パラメータ

`expression:Object` - ActionScript の式またはオブジェクト。

例

この例の `checkEmail()` 関数は、受け取ったストリングが正しいフォーマットの電子メールアドレスかどうかを確認します。ストリングに "@" 記号が含まれていない場合は、エラーをスローします。

```
function checkEmail(email:String) {
    if (email.indexOf "@" == -1) {
        throw new Error("Invalid email address");
    }
}
checkEmail("someuser_theirdomain.com");
```

次に、以下のコードで `try` コードブロック内の `checkEmail()` 関数を呼び出します。`email_txt` ストリングに有効な電子メールアドレスが含まれていない場合は、テキストフィールド `error_txt` にエラーメッセージが表示されます。

```
try {
    checkEmail("Joe Smith");
}
catch (e) {
    error_txt.text = e.toString();
}
```

次の例では、Error クラスのサブクラスをスローします。checkEmail() 関数を修正して、そのサブクラスのインスタンスをスローします。

```
// Define Error subclass InvalidEmailAddress // In InvalidEmailAddress.as: class
InvalidEmailAddress extends Error { var message = "Invalid email address."; }

FLA ファイルまたは AS ファイルで、タイムラインのフレーム 1 に次の ActionScript を入力します。

import InvalidEmailAddress;
function checkEmail(email:String) {
    if (email.indexOf "@" == -1) {
        throw new InvalidEmailAddress();
    }
}
try {
    checkEmail("Joe Smith");
}
catch (e) {
    this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
    error_txt.autoSize = true;
    error_txt.text = e.toString();
}
```

関連項目

[Error](#)

try..catch..finally ステートメント

```
try { ... try block ... } finally { ... finally block ... }
try { ... try block ... }
    catch(error [:ErrorType1]) { ... catch block ... }
    [catch(error[:ErrorTypeN]) { ... catch block ... }]
    [finally { ... finally block ... }]
```

エラーが発生する可能性のあるコードブロックを囲み、そのエラーに対処します。try コードブロック内のコードで throw ステートメントを使用してエラーをスローすると、catch ブロックがある場合は、そのブロックに制御が移ります。さらに finally ブロックがある場合は、そのブロックが実行されます。finally ブロックは、エラーがスローされたかどうかに関係なく、必ず実行されます。try ブロック内のコードがエラーをスローしない場合(try ブロックの実行が正常に終了した場合)でも、finally ブロックのコードは実行されます。return ステートメントを使用して try ブロックを終了した場合でも、finally ブロックは実行されます。

try ブロックの後には、catch ブロックまたは finally ブロック、またはその両方を続ける必要があります。1つの try ブロックに対して複数の catch ブロックを記述することができますが、finally ブロックは1つしか記述できません。try ブロックは、必要なだけ何レベルでもネストできます。

`catch` ハンドラで指定する `error` パラメータには、`e`、`theException`、または `x` のような単純な識別子を使用します。`catch` ハンドラの変数は、タイプ付きにすることもできます。複数の `catch` ブロックを使用する場合は、タイプ付きエラーを使用することにより、1つの `try` ブロックからスローされた複数のタイプのエラーをキャッチすることができます。

スローした例外がオブジェクトである場合、スローしたオブジェクトが指定したタイプのサブクラスであれば、タイプが一致します。特定のタイプのエラーをスローした場合は、対応するエラーを処理する `catch` ブロックが実行されます。指定したタイプではない例外がスローされた場合は、`catch` ブロックは実行されず、`try` ブロックからそのエラーに一致する外部の `catch` ハンドラに対して例外が自動的にスローされます。

関数の内部でエラーをスローし、その関数に `catch` ハンドラが含まれていない場合、その関数は終了します。`catch` ブロックが見つからない限り、呼び出し元の関数もすべて終了します。このプロセスの間に、すべてのレベルの `finally` ハンドラが呼び出されます。

パラメータ

`error:Object - throw` ステートメントでスローされる式。通常は `Error` クラスまたはそのサブクラスのインスタンスです。

例

次の例では、`try..finally` ステートメントの作成方法を示します。`finally` ブロックのコードは必ず実行されるので、通常は `try` ブロックの実行後に必要な "後処理" を実行する場合に使用します。次の例では、`setInterval()` を使用して 1000 ミリ秒(1秒)ごとに関数を呼び出します。エラーが発生すると、エラーがスローされて `catch` ブロックでキャッチされます。エラーの有無に関係なく、`finally` ブロックは必ず実行されます。`setInterval()` が使用されているので、`finally` ブロックに `clearInterval()` を配置して、メモリから間隔を消去する必要があります。

```
myFunction = function () {
    trace("this is myFunction");
};

try {
    myInterval = setInterval(this, "myFunction", 1000);
    throw new Error("my error");
}
catch (myError:Error) {
    trace("error caught: "+myError);
}
finally {
    clearInterval(myInterval);
    trace("error is cleared");
}
```

次の例では、エラーの有無にかかわらず、`finally` ブロックを使用して ActionScript オブジェクトを削除します。`"Account.as"` という名前で新しい AS ファイルを作成します。

```
class Account {  
    var balance:Number = 1000;  
    function getAccountInfo():Number {  
        return (Math.round(Math.random() * 10) % 2);  
    }  
}
```

`"Account.as"` と同じディレクトリに新しい AS ドキュメントまたは FLA ドキュメントを作成し、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
import Account;  
var account:Account = new Account();  
try {  
    var returnVal = account.getAccountInfo();  
    if (returnVal != 0) {  
        throw new Error("Error getting account information.");  
    }  
}  
finally {  
    if (account != null) {  
        delete account;  
    }  
}
```

これは `try..catch` ステートメントの使用例です。`try` ブロック内のコードが実行されます。`try` ブロック内のコードで例外がスローされると、制御が `catch` ブロックに移ります。それにより、`Error.toString()` メソッドを使ってテキストフィールドにエラーメッセージが表示されます。

`"Account.as"` と同じディレクトリに新しい FLA ドキュメントを作成し、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
import Account;  
var account:Account = new Account();  
try {  
    var returnVal = account.getAccountInfo();  
    if (returnVal != 0) {  
        throw new Error("Error getting account information.");  
    }  
    trace("success");  
}  
catch (e) {  
    this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100, 22);  
    status_txt.autoSize = true;  
    status_txt.text = e.toString();  
}
```

次の例では、1つの try コードブロックに対して複数のタイプ付き catch コードブロックを使用します。try ブロックでは、発生したエラーのタイプに応じてさまざまなタイプのオブジェクトをスローします。この例の myRecordSet は、RecordSet という（架空の）クラスのインスタンスです。このクラスの sortRows() メソッドは RecordSetException と MalformedRecord という 2 種類のエラーをスローする可能性があります。

次の例において、RecordSetException オブジェクトと MalformedRecord オブジェクトは Error クラスのサブクラスです。それぞれが独自の AS クラスファイルで定義されています。

```
// In RecordSetException.as:  
class RecordSetException extends Error {  
    var message = "Record set exception occurred.";  
}  
  
// In MalformedRecord.as:  
class MalformedRecord extends Error {  
    var message = "Malformed record exception occurred.";  
}
```

RecordSet クラスの sortRows() メソッドでは、発生した例外のタイプに応じて、これらの定義済みオブジェクトのいずれかをスローします。次に例を示します。

```
class RecordSet {  
    function sortRows() {  
        var returnVal:Number = randomNum();  
        if (returnVal == 1) {  
            throw new RecordSetException();  
        }  
        else if (returnVal == 2) {  
            throw new MalformedRecord();  
        }  
    }  
    function randomNum():Number {  
        return Math.round(Math.random() * 10) % 3;  
    }  
}
```

最後に、他の AS ファイルまたは FLA スクリプトでは、次のコードを使って、RecordSet クラスのインスタンスの sortRows() メソッドを呼び出します。このコードでは、sortRows() でスローされるエラーのタイプごとに、catch ブロックを定義します。

```
import RecordSet;  
var myRecordSet:RecordSet = new RecordSet();  
try {  
    myRecordSet.sortRows();  
    trace("everything is fine");  
}  
catch (e:RecordSetException) {  
    trace(e.toString());  
}  
catch (e:MalformedRecord) {  
    trace(e.toString());  
}
```

関連項目

[Error](#)

var ステートメント

```
var variableName [= value1][...,variableNameN [= valueN]]
```

ローカル変数を宣言する場合に使用します。関数内で変数を宣言した場合、その変数はローカルです。変数はその関数用に定義され、関数呼び出しの終了時にスコープから外れます。具体的には、var を使用して定義された変数は、その変数が含まれるコードブロックでのみ有効です。コードブロックは中括弧 {} で囲まれます。

関数外で変数を宣言した場合、その変数は、そのステートメントが含まれるタイムライン全体で使用できます。

別のオブジェクトにスコープがまたがる変数をローカル変数で宣言することはできません。

```
my_array.length = 25; // ok  
var my_array.length = 25; // syntax error
```

var を使用すると、変数を厳密に型指定することができます。

カンマで宣言を区切ることにより、1つのステートメントで複数の変数を宣言できます。ただし、このシンタックスを使用すると、コードの可読性が悪くなる場合があります。

```
var first:String = "Bart", middle:String = "J.", last:String = "Bartleby";
```

メモ : 外部スクリプトのクラス定義の中でプロパティを宣言する際にも、var を使用する必要があります。クラスファイルでは、変数のスコープとしてパブリック、プライベート、スタティックがサポートされます。

パラメータ

variableName:String - 識別子。

例

次の ActionScript では、製品名の新しい配列を作成します。Array.push を使用して、配列の末尾にエレメントを追加します。厳密な型指定を使用する場合は、var キーワードを使用する必要があります。product_array の前に var を使用しないで厳密な型指定を使おうとすると、エラーが発生します。

```
var product_array:Array = new Array("MX 2004", "Studio", "Dreamweaver", "Flash",  
    "ColdFusion", "Contribute", "Breeze");  
product_array.push("Flex");  
trace(product_array);  
// output: MX 2004,Studio,Dreamweaver,Flash,ColdFusion,Contribute,Breeze,Flex
```

while ステートメント

```
while(condition) { statement(s); }
```

条件を評価して、条件の評価結果が `true` になる場合はステートメントを実行します。その後、ループの先頭に戻り、再び条件を評価します。条件が `false` に評価された場合、ステートメントはスキップされ、ループが終了します。

`while` ステートメントは、次の手順を実行します。手順1～4の各繰り返しはループの反復と呼ばれます。`condition` は、次の手順に示すように、各反復の始めに再テストされます。

- 式 `condition` が評価されます。
- `condition` の評価が `true` であるか、ブール値 `true` に変換される値(ゼロ以外の数値など)である場合は、手順3に進みます。それ以外の場合は、`while` ステートメントが完了し、`while` ループの直後のステートメントから実行が再開されます。
- ステートメントブロック `statement(s)` を実行します。
- 手順1に進みます。

一般にループ処理は、カウンタ変数が指定値より小さい間はアクションを実行するという場合に使用します。各ループの最後で、指定された値に達するまでカウンタがインクリメントされます。指定された値に達すると、`condition` は `true` でなくなり、ループは終了します。

`while` ステートメントによって実行するステートメントブロックを囲む中括弧(())は、実行するステートメントが1つしかない場合は必要ありません。

パラメータ

`condition: Boolean` - 評価結果が `true` または `false` になる式。

例

次の例では、`while` ステートメントを使用して式をテストします。`i` の値が 20 未満の場合に、`i` の値がトレースされます。条件が `true` でなくなると、ループは終了します。

```
var i:Number = 0;
while (i < 20) {
    trace(i);
    i += 3;
}
```

次の結果が [出力] パネルに表示されます。

```
0
3
6
9
12
15
18
```

関連項目

[continue ステートメント](#)

with ステートメント

```
with (object:Object) { statement(s); }
```

object パラメータでムービークリップなどのオブジェクトを指定し、そのオブジェクト内の式やアクションを *statement(s)* パラメータで評価できるようにします。これにより、オブジェクトの名前やパスをオブジェクトに繰り返し書き込む必要がなくなります。

object パラメータは、*statement(s)* パラメータのプロパティ、変数、および関数を読み取るときのコンテキストになります。たとえば、*object* が *my_array* であり、指定されたプロパティのうちの 2つが *length* と *concat* である場合、これらのプロパティは *my_array.length* および *my_array.concat* として自動的に読み取られます。別の例で、*object* が *state.california* である場合、with ステートメント内のアクションまたはステートメントは *california* インスタンス内から呼び出されます。

statement(s) パラメータで識別子の値を検索する場合、ActionScript は *object* で指定されたスコープチェーンの先頭から開始し、特定の順序でスコープチェーンの各レベルで識別子を検索します。識別子を解決するために with ステートメントで使用されるスコープチェーンは、次のようにリストの最初の項目から始まり、最後の項目まで続きます。

- 最も内側の with ステートメントで、*object* パラメータで指定されたオブジェクト。
- 最も外側の with ステートメントで、*object* パラメータで指定されたオブジェクト。
- Activation オブジェクト（ローカル変数を保持する関数が関数内で呼び出されたときに自動的に作成されるテンポラリオブジェクト）
- 実行中のスクリプトを含むムービークリップ
- Global オブジェクト（Math や String などの定義済みオブジェクト）

with ステートメント内に変数を設定するには、with ステートメントの外側で変数を宣言しておくか、変数を設定するタイムラインへのフルパスを入力する必要があります。変数を宣言せずに with ステートメントに設定すると、with ステートメントはスコープチェーンに従って値を検索します。変数がまだ存在しない場合、with ステートメントが呼び出されたタイムライン上に新しい値が設定されます。

with() を使用する代わりに、直接パスを使用することができます。パスが長く、入力が面倒な場合は、ローカル変数を作成してパスを格納しておき、コード内でその変数を再利用することができます。次の ActionScript に例を示します。

```
var shortcut = this._parent._parent.name_txt; shortcut.text = "Hank";
shortcut.autoSize = true;
```

パラメータ

object:Object - ActionScript オブジェクトまたはムービークリップのインスタンス。

例

次の例では、someOther_mc インスタンスの _x プロパティと _y プロパティを設定してから、フレーム 3 に進んで停止するように someOther_mc に指示します。

```
with (someOther_mc) {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```

次のコードの抜粋では、with ステートメントを使用しないで上記コードを記述する方法を示します。

```
someOther_mc._x = 50;  
someOther_mc._y = 100;  
someOther_mc.gotoAndStop(3);
```

with ステートメントは、スコープチェーンリスト内の複数の項目に同時にアクセスする場合に便利です。次の例では、ビルトインの Math オブジェクトをスコープチェーンの前に設定します。Math をデフォルトオブジェクトとして設定すると、cos、sin、PI の各識別子がそれぞれ Math.cos、Math.sin、Math.PI というように解決されます。a、x、y、r の各識別子は Math オブジェクトのメソッドやプロパティではありませんが、関数 polar() のオブジェクトアクティベーションスコープ内に存在するので、それぞれ対応するローカル変数として処理されます。

```
function polar(r:Number):Void {  
    var a:Number, x:Number, y:Number;  
    with (Math) {  
        a = PI * pow(r, 2);  
        x = r * cos(PI);  
        y = r * sin(PI / 2);  
    }  
    trace("area = " + a);  
    trace("x = " + x);  
    trace("y = " + y);  
} polar(3);
```

次の結果が [出力] パネルに表示されます。

```
area = 28.2743338823081  
x = -3  
y = 3
```

fscommand2 コマンド

次のコマンドは、`fscommand2()` 関数で使用できます。`fscommand2()` 関数の詳細については、「グローバル関数」の下にある「[fscommand2 関数](#)」を参照してください。

fscommand2 コマンド

コマンド	説明
<code>ExtendBacklightDuration</code>	バックライトの継続時間を指定した時間だけ延長します。
<code>FullScreen</code>	レンダリングに使用する表示領域のサイズを設定します。
<code>GetBatteryLevel</code>	現在のバッテリーレベルを返します。
<code>GetDevice</code>	Flash Lite が実行されているデバイスを識別するパラメータを設定します。
<code>GetDeviceID</code>	デバイスの一意の識別子(シリアル番号など)を表すパラメータを設定します。
<code>GetFreePlayerMemory</code>	Flash Lite が現在使用できるヒープメモリの量をキロバイトで返します。
<code>GetMaxBatteryLevel</code>	デバイスの最大バッテリーレベルを返します。
<code>GetMaxSignalLevel</code>	現在の最大の信号の強さを数値として返します。
<code>GetMaxVolumeLevel</code>	デバイスの最大音量レベルを数値で返します。
<code>GetNetworkConnectionName</code>	アクティブなネットワーク接続またはデフォルトのネットワーク接続の名前を返します。
<code>GetNetworkConnectStatus</code>	現在のネットワーク接続ステータスを示す値を返します。
<code>GetNetworkGeneration</code>	現在のモバイルワイヤレスネットワークの世代(2G、モバイルワイヤレスの第2世代など)を返します。
<code>GetNetworkName</code>	パラメータを現在のネットワーク名に設定します。
<code>GetNetworkRequestStatus</code>	最新の HTTP 要求のステータスを示す値を返します。
<code>GetNetworkStatus</code>	携帯端末のネットワークステータス(ネットワークが登録されているか、携帯端末が現在ローミング中かどうか)を示す値を返します。
<code>GetPlatform</code>	現在のプラットフォーム(広い意味ではデバイスのクラス)を識別するパラメータを設定します。
<code>GetPowerSource</code>	現在、バッテリーを電源としているか、外部電源を使用しているかを示す値を返します。
<code>GetSignalLevel</code>	現在の信号の強さを数値で返します。
<code>GetSoftKeyLocation</code>	デバイス上のソフトキーの場所を示す値を返します。

コマンド	説明
GetTotalPlayerMemory	Flash Lite に割り当てられたヒープメモリの合計量をキロバイトで返します。
GetVolumeLevel	デバイスの現在の音量レベルを数値で返します。
Quit	Flash Lite プレーヤーの再生を停止して終了します。
ResetSoftKeys	ソフトキーを元の設定にリセットします。
SetFocusRectColor	フォーカス矩形のカラーを自由にできます。
SetInputTextType	入力テキストフィールドを開く際のモードを指定します。
SetSoftKeys	モバイルデバイスのソフトキーをマッピングし直します。
StartVibrate	携帯端末のバイブレータ機能を作動させます。
StopVibrate	バイブレータが動作している場合に停止します。

ExtendBacklightDuration fscommand2 コマンド

ExtendBacklightDuration

バックライトの継続時間を指定した時間だけ延長します。

継続時間が 0 より大きい場合、バックライトをオンにしておくと時間を秒単位（最大 60 秒）で指定します。この時間が経過し、このコマンドが追加で呼び出されなかった場合、バックライトの動作時間がデフォルトの継続時間に戻ります。継続時間が 0 の場合、バックライトの動作は、すぐにデフォルトの動作に戻ります。

メモ：この機能は、システムに依存しています。たとえば、バックライトの最大延長時間が制限されているシステムもあります。

メモ：このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
ExtendBacklightDuration	duration バックライト継続時間（秒）。最大値は 60 秒です。	-1: サポートされない。0: エラーが発生して、操作を完了できなかった。1: 成功。

例

次の例では、バックライトの継続時間を 45 秒間延長します。

```
status = FSCommand2("ExtendBacklightDuration", 45)
```

FullScreen fscommand2 コマンド

FullScreen

レンダリングに使用する表示領域のサイズを設定します。

サイズは、`true`(全画面表示)または`false`(全画面表示より小さい)のいずれかの値を持つ定義済みの変数か定数ストリング値です。これ以外の値は`false`値と見なされます。

メモ : このコマンドは、Flash Lite がスタンドアローンモードで実行されている場合のみサポートされています。Flash Lite プレーヤーが他のアプリケーションのコンテキストで実行されている場合(プラウザのプラグインとして実行されている場合など)、このコマンドは使用できません。

コマンド	パラメータ	戻り値
FullScreen	size	-1: サポートされない。0: サポートされる。

例

次の例では、表示領域のサイズを全画面表示に設定します。

```
status = fscommand2("FullScreen", true);
```

GetBatteryLevel fscommand2 コマンド

GetBatteryLevel

現在のバッテリーレベルを返します。返される値は、0 から `GetMaxBatteryLevel()` が返す最大値の間の数値です。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetBatteryLevel	なし	-1: サポートされない。その他の数値: 現在のバッテリー容量。

例

次の例では、`battLevel` 変数を現在のバッテリーの容量に設定します。

```
battLevel = fscommand2("GetBatteryLevel");
```

GetDevice fscommand2 コマンド

GetDevice

Flash Lite が実行されているデバイスを識別するパラメータを設定します。この ID は、通常、モデル名です。

コマンド	パラメータ	戻り値
GetDevice	device デバイスの ID を受け取るストリング。これは変数の名前、または変数の名前が含まれるストリング値です。	-1: サポートされない。0: サポートされる。

例

次の例ではデバイスの ID を device に代入します。

```
status = fscommand2("GetDevice", "device");
```

次に結果のサンプルとそれが表すデバイスを示します。

D506i Mitsubishi 506i 携帯端末。DFOMA1 Mitsubishi FOMA1 携帯端末。F506i Fujitsu 506i 携帯端末。FFOMA1 Fujitsu FOMA1 携帯端末。N506i NEC 506i 携帯端末。NFOMA1 NEC FOMA1 携帯端末。Nokia3650 Nokia 3650 携帯端末。p506i Panasonic 携帯端末。PFOMA1 Panasonic FOMA1 携帯端末。SH506i Sharp 506i 携帯端末。SHFOMA1 Sharp FOMA1 携帯端末。S0506i Sony 506i 携帯端末。

GetDeviceID fscommand2 コマンド

GetDeviceID

デバイスの一意の識別子(シリアル番号など)を表すパラメータを設定します。

コマンド	パラメータ	戻り値
GetDeviceID	id デバイスの一意な ID を受け取るストリング。これは変数の名前、または変数の名前が含まれるストリング値です。	-1: サポートされない。0: サポートされる。

例

次の例では、一意な ID を変数 deviceId に代入します。

```
status = fscommand2("GetDeviceID", "deviceId");
```

GetFreePlayerMemory fscommand2 コマンド

GetFreePlayerMemory

Flash Lite が現在使用できるヒープメモリの量をキロバイトで返します。

コマンド	パラメータ	戻り値
GetFreePlayerMemory	なし	-1: サポートされない。0 または正の値： 使用可能なヒープメモリのキロバイト数。

例

次の例では、ステータスを使用可能なメモリと等しい量に設定します。

```
status = fscommand2("GetFreePlayerMemory");
```

GetMaxBatteryLevel fscommand2 コマンド

GetMaxBatteryLevel

デバイスの最大バッテリーレベルを返します。返される値は、正の数値です。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetMaxBatteryLevel	なし	-1: サポートされない。その他の値：最大 バッテリー容量。

例

次の例では、maxBatt 変数を最大バッテリーレベルに設定します。

```
maxBatt = fscommand2("GetMaxBatteryLevel");
```

GetMaxSignalLevel fscommand2 コマンド

GetMaxSignalLevel

現在の最大の信号の強さを数値として返します。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetMaxSignalLevel	なし	-1: サポートされない。その他の数値： 最大信号レベル

例

次の例では、最大信号強さを変数 sigStrengthMax に代入します。

```
sigStrengthMax = fscommand2("GetMaxSignalLevel");
```

GetMaxVolumeLevel fscommand2 コマンド

GetMaxVolumeLevel

デバイスの最大音量レベルを数値で返します。

コマンド	パラメータ	戻り値
GetMaxVolumeLevel	なし	-1: サポートされない。その他の値: 最大音量レベル

例

次の例では、変数 maxvolume をデバイスの最大音量レベルに設定します。

```
maxvolume = fscommand2("GetMaxVolumeLevel");
trace (maxvolume); // output: 80
```

GetNetworkConnectionName fscommand2 コマンド

GetNetworkConnectionName

アクティブなネットワーク接続またはデフォルトのネットワーク接続の名前を返します。モバイルデバイスの場合、この接続はアクセスポイントとも呼ばれます。

メモ: このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetNetworkConnectionName	なし	-1: サポートされない。0: 成功。アクティブなネットワーク接続を返す。 1: 成功。デフォルトのネットワーク接続を返す。2: 接続名を取得できない。

例

次の例は、myConnectionName 引数でアクティブなまたはデフォルトのネットワーク接続を返します。

```
status = FSCommand2("GetNetworkConnectionName", "myConnectionName");
```

GetNetworkConnectStatus fscommand2 コマンド

GetNetworkConnectStatus

現在のネットワーク接続ステータスを示す値を返します。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetNetworkConnectStatus	なし	-1: サポートされない。0: 現在アクティブなネットワーク接続がある。 1: デバイスがネットワークへの接続を試行中。2: 現在アクティブなネットワーク接続がない。3: ネットワーク接続が中断している。4: ネットワーク接続を確認できない。

例

次の例では、ネットワーク接続ステータスを変数 connectstatus に代入し、switch ステートメントを使用して接続のステータスでテキストフィールドを更新します。

```
connectstatus = FSCommand2("GetNetworkConnectStatus");
switch (connectstatus) {
connectstatus = FSCommand2("GetNetworkConnectStatus");
switch (connectstatus) {
connectstatus = FSCommand2("GetNetworkConnectStatus");
switch (connectstatus) {
case -1 :
_root.myText += "connectstatus not supported" + "\n";
break;
case 0 :
_root.myText += "connectstatus shows active connection" + "\n";
break;
case 1 :
_root.myText += "connectstatus shows attempting connection" + "\n";
break;
case 2 :
_root.myText += "connectstatus shows no connection" + "\n";
break;
case 3 :
_root.myText += "connectstatus shows suspended connection" + "\n";
break;
case 4 :
_root.myText += "connectstatus shows indeterminable state" + "\n";
break;
}
```

GetNetworkGeneration fscommand2 コマンド

GetNetworkGeneration

現在のモバイルワイヤレスネットワークの世代(2G、モバイルワイヤレスの第2世代など)を返します。

コマンド	パラメータ	戻り値
GetNetworkGeneration	なし	-1:サポートされない。0:モバイルワイヤレスネットワークの不明の世代。 1:2G 2:2.5G 3:3G

例

次の例では、ネットワークの世代を返すためのシンタックスを示します。

```
status = fscommand2("GetNetworkGeneration");
```

GetNetworkName fscommand2 コマンド

GetNetworkName

パラメータを現在のネットワーク名に設定します。

メモ: このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetNetworkName	networkName ネットワーク名を表すストリング。これは変数の名前、または変数の名前が含まれるストリング値です。ネットワークが登録され、ネットワーク名が確認できる場合、networkname はネットワーク名に設定されます。確認できない場合は空のストリングに設定されます。	-1:サポートされない。0:ネットワークが登録されていない。1:ネットワークが登録されているが、ネットワーク名がわからない。2:ネットワークが登録されていて、ネットワーク名がわかる。

例

次の例では、現在のネットワークの名前をパラメータ myNetName に代入し、ステータス値を変数 netNameStatus に代入します。

```
netNameStatus = fscommand2("GetNetworkName", myNetName);
```

GetNetworkRequestStatus fscommand2 コマンド

GetNetworkRequestStatus

最新の HTTP 要求のステータスを示す値を返します。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetNetworkRequestStatus	なし	-1: コマンドはサポートされない。0: 保留中の要求があり、ネットワーク接続が確立され、サーバーのホスト名が解決されており、サーバーに接続されている。1: 保留中の要求があり、ネットワーク接続を確立中。2: 保留中の要求があるが、ネットワーク接続がまだ確立されていない。3: 保留中の要求があり、ネットワーク接続が確立され、サーバーのホスト名を解決中。4: ネットワークエラーにより要求が失敗した。5: サーバーへの接続エラーにより要求が失敗した。6: サーバーから HTTP エラー (404 など) が返された。7: DNS サーバーにアクセスできないかサーバー名を解決できないために要求が失敗した。8: 要求が正常に完了した。9: タイムアウトにより要求が失敗した。10: まだ要求が発行されていない。

例

次の例では、直近の HTTP 要求のステータスを変数 requeststatus に代入し、switch ステートメントを使用して接続のステータスでテキストフィールドを更新します。

```
requeststatus = fscommand2("GetNetworkRequestStatus");
switch (requeststatus) {
    case -1:
        _root.myText += "requeststatus not supported" + "\n";
        break;
    case 0:
        _root.myText += "connection to server has been made" + "\n";
        break;
    case 1:
        _root.myText += "connection is being established" + "\n";
        break;
    case 2:
        _root.myText += "pending request, contacting network" + "\n";
        break;
```

```

case 3:
_root.myText += "pending request, resolving domain" + "\n";
break;
case 4:
_root.myText += "failed, network error" + "\n";
break;
case 5:
_root.myText += "failed, couldn't reach server" + "\n";
break;
case 6:
_root.myText += "HTTP error" + "\n";
break;
case 7:
_root.myText += "DNS failure" + "\n";
break;
case 8:
_root.myText += "request has been fulfilled" + "\n";
break;
case 9:
_root.myText += "request timedout" + "\n";
break;
case 10:
_root.myText += "no HTTP request has been made" + "\n";
break;
}

```

GetNetworkStatus fscommand2 コマンド

GetNetworkStatus

携帯端末のネットワークステータス(ネットワークが登録されているか、携帯端末が現在ローミング中かどうか)を示す値を返します。

コマンド	パラメータ	戻り値
GetNetworkStatus	なし	-1: コマンドはサポートされない。0: ネットワークが登録されていない。1: ホームネットワークに接続されている。2: 拡張ホームネットワークに接続されている。3: ローミング中(ホームネットワーク以外に接続)。

例

次の例では、ネットワーク接続のステータスを変数 networkstatus に代入し、switch ステートメントを使用してステータスでテキストフィールドを更新します。

```

networkstatus = fscommand2("GetNetworkStatus");
switch(networkstatus) {
case -1:
_root.myText += "network status not supported" + "\n";

```

```

break;
case 0:
_root.myText += "no network registered" + "\n";
break;
case 1:
_root.myText += "on home network" + "\n";
break;
case 2:
_root.myText += "on extended home network" + "\n";
break;
case 3:
_root.myText += "roaming" + "\n";
break;
}

```

GetPlatform fscommand2 コマンド

GetPlatform

現在のプラットフォーム(広い意味ではデバイスのクラス)を識別するパラメータを設定します。オープンなオペレーティングシステムを使用しているデバイスの場合、通常はオペレーティングシステムの名前とバージョンが識別されます。

コマンド	パラメータ	戻り値
GetPlatform	platform プラットフォームの ID を受け取るストリング。	-1: サポートされない。0: サポートされる。

例

次の例では、platform パラメータに現在のプラットフォームの識別子を設定します。

```
status = fscommand2("GetPlatform", "platform");
```

次の例では、platform の結果のサンプルを示します。

506i 506i 携帯端末。FOMA1 FOMA1 携帯端末。Symbian6.1_s60.1 Symbian 6.1、Series 60 バージョン1 携帯端末。Symbian7.0 Symbian 7.0 携帯端末。

GetPowerSource fscommand2 コマンド

GetPowerSource

現在、バッテリーを電源としているか、外部電源を使用しているかを示す値を返します。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetPowerSource	なし	-1: サポートされない。0: デバイスはバッテリー電源で動作している。1: デバイスは外部電源で動作している。

例

次の例では、電源を示すように変数 myPower を設定します。設定できない場合は、-1 に設定します。

```
myPower = fscommand2("GetPowerSource");
```

GetSignalLevel fscommand2 コマンド

GetSignalLevel

現在の信号の強さを数値で返します。

メモ : このコマンドは、BREW デバイスではサポートされていません。

コマンド	パラメータ	戻り値
GetSignalLevel	なし	-1: サポートされない。その他の数値：現在の信号レベル(0 から GetMaxSignalLevel が返す最大値の範囲)。

例

次の例では、信号レベルの値を変数 sigLevel に代入します。

```
sigLevel = fscommand2("GetSignalLevel");
```

GetSoftKeyLocation fscommand2 コマンド

GetSoftKeyLocation

デバイス上のソフトキーの場所を示す値を返します。

コマンド	パラメータ	戻り値
GetSoftKeyLocation	なし	-1: サポートされない。0: ソフトキーがトップ。1: ソフトキーが左側。2: ソフトキーが下部。3: ソフトキーが右側。

例

次は、status 変数を設定してソフトキーの場所を示す、またはソフトキーがデバイスでサポートされていない場合に変数を -1 に設定する例です。

```
status = fscommand2("GetSoftKeyLocation");
```

GetTotalPlayerMemory fscommand2 コマンド

GetTotalPlayerMemory

Flash Lite に割り当てられたヒープメモリの合計量をキロバイトで返します。

コマンド	パラメータ	戻り値
GetTotalPlayerMemory	なし	-1: サポートされない。0 または正の値: ヒープメモリの総量(キロバイト単位)

例

次の例では、status 変数をヒープメモリの総量に設定します。

```
status = fscommand2("GetTotalPlayerMemory");
```

GetVolumeLevel fscommand2 コマンド

GetVolumeLevel

デバイスの現在の音量レベルを数値で返します。

コマンド	パラメータ	戻り値
GetVolumeLevel	なし	-1: サポートされない。その他の数値: 現在の音量レベル(0 から fscommand2("GetMaxVolumeLevel") が返す値の範囲)。

例

次の例では、現在の音量レベルを変数 volume に代入します。

```
volume = fscommand2("GetVolumeLevel");
trace (volume); // output: 50
```

Quit fscommand2 コマンド

Quit

Flash Lite プレーヤーの再生を停止して終了します。

このコマンドは、Flash Lite がスタンドアローンモードで実行されている場合のみサポートされています。Flash Lite プレーヤーが他のアプリケーションのコンテキストで実行されている場合(ブラウザのプラグインとして実行されている場合など)、このコマンドは使用できません。

コマンド	パラメータ	戻り値
Quit	なし	-1: サポートされない。

例

次の例では、スタンドアローンモードで実行中に Flash Lite の再生を停止して終了します。

```
status = fscommand2("Quit");
```

ResetSoftKeys fscommand2 コマンド

ResetSoftKeys

ソフトキーを元の設定にリセットします。

このコマンドは、Flash Lite がスタンダードアローンモードで実行されている場合のみサポートされています。Flash Lite プレーヤーが他のアプリケーションのコンテキストで実行されている場合（ブラウザのプラグインとして実行されている場合など）、このコマンドは使用できません。

コマンド	パラメータ	戻り値
ResetSoftKeys	なし	-1: サポートされない。

例

次のステートメントは、ソフトキーを元の設定にリセットします。

```
status = fscommand2("ResetSoftKeys");
```

SetFocusRectColor fscommand2 コマンド

SetFocusRectColor

フォーカス矩形のカラーを自由にできます。

赤、緑、青に対して指定できるカラーの範囲は 0 ~ 255 です。Flash の場合は、フォーカス矩形のデフォルトのカラー（黄色）は変更できません。

コマンド	パラメータ	戻り値
SetFocusRectColor	なし	-1: サポートされない。0: 確定できない。 1: 成功。

例

次のステートメントは、フォーカス矩形のカラーをリセットします。

```
status = fscommand2("SetFocusRectColor, <red>, <green>, <blue>);
```

SetInputTextType fscommand2 コマンド

SetInputTextType

入力テキストフィールドを開く際のモードを指定します。

Flash Lite では、ホストアプリケーションに汎用のデバイス依存テキスト入力インターフェイスを開始するよう要求することで、入力テキスト機能をサポートします。これは通常 FEP (Front-End Processor : 前置プロセッサ) と呼ばれます。SetInputTextType コマンドを使用しない場合は、FEP がデフォルトモードで開きます。

コマンド	パラメータ	戻り値
SetInputTextType	variableName 入力テキストフィールドの名前。これは変数の名前、または変数の名前が含まれるストリング値です。 メモ: テキストフィールドの変数名は、インスタンス名と同じではありません。テキストフィールドの変数名を、プロパティインスペクタの [変数] テキストボックスに割り当てるか、または ActionScript を使用して割り当てることができます。たとえば、次のコードは、変数名 "numTxt_var" が関連付けられているテキストフィールドインスタンス (numTxt) への数値文字の入力を制限します。 var numTxt:TextField;numTxt.variable = "numTxt_var";fscommand2("SetInputTextType", "numTxt_var", "Numeric"); type Numeric、Alpha、Alphanumeric、Latin、NonLatin、または NoRestriction のいずれかの値。	0 : 失敗。1 : 成功。

下表に、各モードによる影響と置換されるモードを示します。

InputTextType モード	FEP をこれらの相互に排他的なモードのいずれかに設定する	現在のデバイスでサポートされていない場合は、このモードで FEP を開く
Numeric	数値のみ (0 ~ 9)	Alphanumeric
Alpha	アルファベット文字のみ (A ~ Z, a ~ z)	Alphanumeric
Alphanumeric	英数字のみ (0 ~ 9, A ~ Z, a ~ z)	Latin
Latin	ラテン文字のみ (英数字と句読記号)	NoRestriction
NonLatin	非ラテン文字のみ (漢字とかななど)	NoRestriction
NoRestriction	デフォルトのモード (FEP に制限を設定しない)	該当なし

メモ:すべての携帯端末でこれらの入力テキストフィールドの種類がサポートされているとは限らないので、入力されるテキストデータは検証する必要があります。

例

次のコードは、数値データを受け取るために、変数 input1 に関連付けられたフィールドのテキストの種類を設定します。

```
status = fscommand2("SetInputTextType", "input1", "Numeric");
```

SetSoftKeys fscommand2 コマンド

SetSoftKeys

モバイルデバイスのソフトキーをマッピングし直します。

ユーザーがソフトキーを押すと、ソフトキーイベントに関連付けられている ActionScript が実行されます。Flash Lite プレーヤーは、起動されると直ちにこの関数を実行します。このコマンドは、Flash Lite がスタンダードアローンモードで実行されている場合のみサポートされています。Flash Lite プレーヤーが他のアプリケーションのコンテキストで実行されている場合(ブラウザのプラグインとして実行されている場合など)、このコマンドは使用できません。

Flash Lite 1.1との後方互換性のために、SOFT1 ソフトキーは、常に携帯端末の左のキーにマッピングされ、SOFT2 ソフトキーは常に携帯端末の右のキーにマッピングされます。SOFT3 ソフトキーおよびそれ以降のソフトキーの位置は、各携帯端末によって異なります。

このコマンドの引数では、対応するソフトキーに対して表示されるテキストを指定します。SetSoftKeys コマンドの実行後、左のキーを押すと SOFT1 キー押下イベントが発生し、右のキーを押すと SOFT2 キー押下イベントが発生します。SOFT3 から SOFT12 までのソフトキーを押すと、それぞれ対応するイベントが発生します。

メモ: ソフトキーの再マッピングは、モバイルデバイスに依存します。ソフトキーの再マッピングがサポートされているかどうかについては、デバイスの製造元に確認してください。

コマンド	パラメータ	戻り値
SetSoftKeys	soft1 SOFT1 ソフトキーに表示するテキスト。soft2 SOFT2 ソフトキーに表示するテキスト。 これらのパラメータは変数の名前または定数ストリング値です(たとえば、Previous)。	-1: サポートされない。0: サポートされる。

例

次の例では、SOFT1 ソフトキーに "Previous"、SOFT2 ソフトキーに "Next" と表示します。

```
status = fscommand2("SetSoftKeys", "Previous", "Next");
```

各ソフトキーに対して変数を定義するか、定数ストリング値を使用することができます。

```
status = fscommand2("SetSoftKeys", soft1, soft2, [soft3], [soft4], ..., [softn])
```

メモ: 他のソフトキーを設定せずに1つのソフトキーを設定することができます。次の例では、他のキーに影響を与える前に特定のソフトキーを設定する場合のシンタックスと動作を示します。

- 左のソフトキーのラベルを "soft1" に設定し、右のソフトキーのラベルを空白に設定するには、次のコマンドを使用します。

```
status = fscommand2("SetSoftKeys", "soft1", "")
```

- 左のソフトキーのラベルをそのままにして、右のソフトキーのラベルを "soft2" に設定するには、次のコマンドを使用します。

```
status = fscommand2("SetSoftKeys", undefined, "soft2")
```

- 左のソフトキーのラベルをそのままにして、右のソフトキーのラベルを "soft2" に設定するには、次のコマンドを使用します。

```
status = fscommand2("SetSoftKeys", null, "soft2")
```

- 左のソフトキーのラベルを "soft1" に設定し、右のソフトキーをそのままにするには、次のコマンドを使用します。

```
status = fscommand2("SetSoftKeys", "soft1")
```

StartVibrate fscommand2 コマンド

StartVibrate

携帯端末のバイブレータ機能を作動させます。

バイブレータが既に動作している場合は、Flash Lite は現在の動作を停止してから、新しく指定されたバイブレータ動作を開始します。Flash アプリケーションの再生を停止または一時停止したとき、および Flash Lite プレーヤーを終了したときにもバイブレータは停止します。

コマンド	パラメータ	戻り値
StartVibrate	time_on バイブレータがオンになっているミリ秒単位の期間(最大5秒)。 time_off バイブレータがオフになっているミリ秒単位の期間(最大5秒)。 repeat この動作を繰り返す回数(最大3回)。	-1:サポートされない。0:バイブレーションが開始した。1:エラーが発生してバイブレーションを開始できなかった。

例

次の例では、2.5秒間オン、1秒間オフのバイブレーションシーケンスを開始して2回繰り返します。そして、成否を示す変数 status に値を代入します。

```
fscommand2("StartVibrate", 2500, 1000, 2);
```

StopVibrate fscommand2 コマンド

StopVibrate

バイブレータが動作している場合に停止します。

コマンド	パラメータ	戻り値
StopVibrate	なし	-1:サポートされない。0:バイブレーションが停止した。

例

次の例では、StopVibrate を呼び出して、結果(サポートされない、またはバイブレーションが停止)を status 変数に格納します。

```
status = fscommand2("StopVibrate");
```

第2章

ActionScript クラス

ActionScript クラスのドキュメントには、シンタックスや使用方法に関する情報が含まれているほか、ActionScript の特定のクラスに属するメソッド、プロパティ、イベントハンドラとリスナーのコードサンプルも含まれています（グローバル関数やグローバルプロパティは除く）。クラスはアルファベット順に並んでいます。特定のメソッドまたはプロパティがどのクラスに属しているのかわからない場合、インデックスで調べることができます。

arguments

```
Object
|
+- arguments
```

```
public class arguments
extends Object
```

arguments オブジェクトは、関数の引数を保存したり、引数にアクセスしたりするのに使用されます。関数の本体に含まれる場合、ローカルの arguments 変数を使ってアクセスできます。

引数は配列エレメントとして保存され、最初の引数は arguments[0] として、2 番目の引数は arguments[1] としてアクセスされます。arguments.length プロパティは、関数に渡される引数の数を示します。関数で宣言された数と異なる数の引数が渡される場合があることに注意してください。

関連項目

[Function](#)

プロパティ一覧

オプション	プロパティ	説明
	<code>callee:Object</code>	現在実行中の関数への参照。
	<code>caller:Object</code>	現在実行中の関数を呼び出した関数への参照。別の関数から呼び出された関数でない場合は <code>null</code> を返します。
	<code>length:Number</code>	関数に渡される引数の数。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

メソッド一覧

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

callee (arguments.callee プロパティ)

public callee : Object

現在実行中の関数への参照。

関連項目

[caller \(arguments.caller プロパティ \)](#)

caller (arguments.caller プロパティ)

public caller : Object

現在実行中の関数を呼び出した関数への参照。別の関数から呼び出された関数でない場合は `null` を返します。

関連項目

[callee \(arguments.callee プロパティ \)](#)

length (arguments.length プロパティ)

public length : Number

関数に渡される引数の数。関数で宣言された数よりも増減する場合があります。

Array

Object

|

+ - Array

public dynamic class Array
extends Object

Array クラスを使用すると、インデックス付き配列にアクセスして操作できます。インデックス付き配列とは、配列内での位置を表す番号でプロパティを特定するオブジェクトのことです。この番号のことをインデックスと言います。すべてのインデックス付き配列はゼロから始まります。つまり、配列内の第1のエレメントは [0] であり、第2のエレメントは [1] です(以下同様)。Array オブジェクトを作成するには、コンストラクタ new Array() を使用します。配列の各エレメントにアクセスするには、配列アクセス演算子 ([]) を使用します。

配列エレメントには、数値、ストリング、オブジェクト、他の配列など、各種データ型を保存できます。多次元配列は、インデックス付き配列を作成し、その各エレメントに異なるインデックス配列を代入することで作成できます。このような配列は、表内のデータを表現する場合に使用できるので多次元であると見なされます。

配列では、値による代入ではなく、参照による代入が行われます。ある配列変数に別の配列変数を代入すると、両方とも同じ配列を参照するようになります。

```
var oneArray:Array = new Array("a", "b", "c");
var twoArray:Array = oneArray; // Both array variables refer to the same array.
twoArray[0] = "z";
trace(oneArray); // Output: z,b,c.
```

Array クラスは、連想配列を作成する場合に使用しないでください。連想配列は異なるデータ構造を備えており、数値エレメントではなく、名前エレメントがあります。連想配列(ハッシュとも言います)を作成する場合は、Object クラスを使用してください。ActionScript では Array クラスを使用して連想配列を作成できますが、Array クラスのメソッドやプロパティは使用できません。基本的に、連想配列は Object クラスのインスタンスであり、キー値ペアはプロパティとその値という形式で表現されます。Object 型を使用して連想配列を宣言する別の理由は、そうすると、オブジェクトリテラルを使用して連想配列にデータを設定できるからです(ただし、宣言時のみ)。次の例では、オブジェクトリテラルを使用して連想配列を作成し、ドット演算子と配列アクセス演算子を両方とも使用して項目にアクセスします。その後、新しいプロパティを作成することで新しいキー値ペアを追加します。

```
var myAssocArray:Object = {fname:"John", lname:"Public"};
trace(myAssocArray.fname); // Output: John
```

```

trace(myAssocArray["lname"]); // Output: Public
myAssocArray.initial = "Q";
trace(myAssocArray.initial); // Output: Q

```

例

次の例では、`my_array` を 4 つの月の名前で構成します。

```

var my_array:Array = new Array();
my_array[0] = "January";
my_array[1] = "February";
my_array[2] = "March";
my_array[3] = "April";

```

プロパティ一覧

オプション	プロパティ	説明
static	CASEINSENSITIVE:Number	大文字と小文字を区別しないソートを表します。
static	DESCENDING:Number	降順のソート順を表します。
	length:Number	配列内のエレメント数を示す負でない整数。
static	NUMERIC:Number	ストリングベースのソートではなく、数値によるソートを表します。
static	RETURNINDEXEDARRAY:Number	<code>sort()</code> または <code>sortOn()</code> メソッドの呼び出し結果としてインデックス付き配列を返すオプションを表します。
static	UNIQUESORT:Number	一意性ソート要件を表します。

Object クラスから継承されるプロパティ

constructor (Object. constructor プロパティ), __proto__ (Object. __proto__ プロパティ), prototype (Object. prototype プロパティ), __resolve (Object. __resolve プロパティ)
--

コンストラクター一覧

シグネチャ	説明
<code>Array ([value:Object])</code>	配列を作成できます。

メソッド一覧

オプション	シグネチャ	説明
	<code>concat ([value:Object]) : Array</code>	パラメータで指定されたエレメントを配列内のエレメントと連結して、新しい配列を作成します。
	<code>join ([delimiter:String]) : String</code>	配列内のエレメントをストリングに変換し、指定されたセパレータをエレメント間に挿入し、エレメントを連結して、その結果をストリングとして返します。
	<code>pop() : Object</code>	配列の最後のエレメントを削除して、そのエレメントの値を返します。
	<code>push(value:Object) : Number</code>	エレメントを配列の最後に追加して、配列の新しい長さを返します。
	<code>reverse() : Void</code>	配列の並びを反転させます。
	<code>shift() : Object</code>	配列の最初のエレメントを削除して、そのエレメントを返します。
	<code>slice([startIndex:Number], [endIndex:Number]) : Array</code>	元の配列から一連のエレメントを取り出して、新しい配列を返します。元の配列は変更されません。
	<code>sort([compareFunction:Object], [options:Number]) : Array</code>	配列内のエレメントをソートします。
	<code>sortOn (fieldName:Object, [options:Object]) : Array</code>	配列内のフィールド(複数のフィールドも可能)に基づいて、配列内のエレメントをソートします。
	<code>splice(startIndex:Number, [deleteCount:Number], [value:Object]) : Array</code>	配列のエレメントを追加および削除します。
	<code>toString() : String</code>	指定された Array オブジェクト内のエレメントを表すストリングを返します。
	<code>unshift(value:Object) : Number</code>	エレメントを配列の先頭に追加して、配列の新しい長さを返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

Array コンストラクタ

public Array([value:Object])

配列を作成できます。コンストラクタを使用して各種の配列を作成できます。たとえば、空の配列や、長さのみが指定されておりエレメントの値が定義されていない配列のほか、エレメントが特定の値を持つ配列などがあります。

シンタックス1: パラメータを指定しない場合は、長さが0の配列が作成されます。

シンタックス2: 長さのみを指定した場合は、lengthで指定した数のエレメントを持つ配列が作成されます。各エレメントの値はundefinedに設定されます。

シンタックス3: elementパラメータを使って値を指定した場合は、特定の値を持つ配列が作成されます。

パラメータ

value:Object(オプション)-次のいずれか:

- 配列内のエレメント数を指定する整数。
- 複数の任意の値で構成されるリスト。値として使用できる型は、Boolean、Number、String、Object、Arrayです。配列内の第1のエレメントのインデックス番号は常に0です。

メモ: Arrayコンストラクタに数値パラメータを1つだけ渡すと、そのパラメータはlengthと見なされ、Integer()関数を使用して整数に変換されます。

例

シンタックス1: 次の例では、エレメント数0個の新しいArrayオブジェクトを作成します。

```
var my_array:Array = new Array();  
trace(my_array.length); // Traces 0.
```

シンタックス2: 次の例では、エレメント数4個の新しいArrayオブジェクトを作成します。

```
var my_array:Array = new Array(4);  
trace(my_array.length); // Returns 4.  
trace(my_array[0]); // Returns undefined.  
if (my_array[0] == undefined) { // No quotation marks around undefined.  
    trace("undefined is a special value, not a string");  
} // Traces: undefined is a special value, not a string.
```

シンタックス 3:次の例では、初期のエレメント数が 5 個の新しい Array オブジェクト go_gos_array を作成します。

```
var go_gos_array:Array = new Array("Belinda", "Gina", "Kathy", "Charlotte",
    "Jane");
trace(go_gos_array.length); // Returns 5.
trace(go_gos_array.join(", ")); // Displays elements.
```

go_gos_array 配列の初期エレメントを次のように指定します。

```
go_gos_array[0] = "Belinda";
go_gos_array[1] = "Gina";
go_gos_array[2] = "Kathy";
go_gos_array[3] = "Charlotte";
go_gos_array[4] = "Jane";
```

次のコードでは、go_gos_array 配列に 6 番目のエレメントを追加し、2 番目のエレメントを変更します。

```
go_gos_array[5] = "Donna";
go_gos_array[1] = "Nina"
trace(go_gos_array.join(" + "));
// Returns Belinda + Nina + Kathy + Charlotte + Jane + Donna.
```

関連項目

[\[\] 配列アクセス演算子 , length \(Array.length プロパティ \)](#)

CASEINSENSITIVE (Array.CASEINSENSITIVE プロパティ)

public static CASEINSENSITIVE : Number

大文字と小文字を区別しないソートを表します。この定数は、sort() メソッドまたは sortOn() メソッドの options パラメータで使用できます。この定数の値は 1 です。

関連項目

[sort \(Array.sort メソッド \), sortOn \(Array.sortOn メソッド \)](#)

concat (Array.concat メソッド)

```
public concat([value:Object]) : Array
```

パラメータで指定されたエレメントを配列内のエレメントと連結して、新しい配列を作成します。

value パラメータで配列が指定されている場合は、配列自体ではなく、その配列のエレメントが連結されます。配列 my_array は変更されません。

パラメータ

value: Object (オプション) - 新しい配列内で連結される数値、エレメント、またはストリング。値を渡さない場合には、my_array の複製が作成されます。

戻り値

Array - この配列のエレメントの後にパラメータのエレメントが続く配列。

例

次のコードは、2つの配列を連結します。

```
var alpha_array:Array = new Array("a","b","c");
var numeric_array:Array = new Array(1,2,3);
var alphaNumeric_array:Array =alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// Creates array [a,b,c,1,2,3].
```

次のコードは、3つの配列を連結します。

```
var num1_array:Array = [1,3,5];
var num2_array:Array = [2,4,6];
var num3_array:Array = [7,8,9];
var nums_array:Array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// Creates array [1,3,5,2,4,6,7,8,9].
```

ネストされた配列は、通常の配列と同じように処理されます。ネストされた配列内のエレメントは、配列 x_array 内で個別のエレメントとして分解されません。次に例を示します。

```
var a_array:Array = new Array ("a","b","c");

// 2 and 3 are elements in a nested array.
var n_array:Array = new Array(1, [2, 3], 4);

var x_array:Array = a_array.concat(n_array);
trace(x_array[0]); // a
trace(x_array[1]); // b
trace(x_array[2]); // c
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

DESCENDING (Array.DESCENDING プロパティ)

public static DESCENDING : Number

降順のソート順を表します。この定数は、sort() メソッドまたは sortOn() メソッドの options パラメータで使用できます。この定数の値は 2 です。

関連項目

[sort \(Array.sort メソッド\)](#), [sortOn \(Array.sortOn メソッド\)](#)

join (Array.join メソッド)

public join([delimiter:String]) : String

配列内のエレメントをストリングに変換し、指定されたセパレータをエレメント間に挿入し、エレメントを連結して、その結果をストリングとして返します。ネストされた配列は、join() メソッドに渡されるセパレータで区切るのではなく、常にカンマ(,)で区切れます。

パラメータ

delimiter:String (オプション) - 返されるストリング内の配列エレメントを区切る文字またはストリング。このパラメータを省略すると、デフォルトのセパレータとしてカンマ(,)が使用されます。

戻り値

String - ストリング。

例

次の例では、Earth、Moon および Sun という 3 つのエレメントを含む配列を作成します。次に、配列を 3 回結合します。-1回目はデフォルトのセパレータ(カンマ(,)とスペース)を使い、2回目はダッシュ(-)を使い、最後はプラス記号(+)を使います。

```
var a_array:Array = new Array("Earth","Moon","Sun")
trace(a_array.join());
// Displays Earth,Moon,Sun.
trace(a_array.join(" - "));
// Displays Earth - Moon - Sun.
trace(a_array.join(" + "));
// Displays Earth + Moon + Sun.
```

次の例では、2つの配列がネストされた配列を作成します。1番目の配列には、Europa、Io、および Callisto という3つのエレメントを格納します。2番目の配列には、Titan および Rhea という2つのエレメントを格納します。配列をプラス記号 (+) で結合しても、ネストされた配列内の各エレメントはカンマ (,) で区切られたままになります。

```
var a_nested_array:Array = new Array(["Europa", "Io", "Callisto"], ["Titan",  
    "Rhea"]);  
trace(a_nested_array.join(" + "));  
// Returns Europa,Io,Callisto + Titan,Rhea.
```

関連項目

[split \(String.split メソッド\)](#)

length (Array.length プロパティ)

public length : Number

配列内のエレメント数を示す負でない整数。このプロパティは、新しいエレメントが配列に追加されると自動更新されます。配列エレメントに値を代入するとき (my_array[index] = value など)、index が数値でかつ index+1 が length プロパティよりも大きい場合、length プロパティが index+1 に更新されます。

メモ: length プロパティに既存の長さよりも短い値を代入した場合、配列は切り詰められます。

例

次のコードでは、length プロパティがどのように更新されるかを示します。最初の長さが0で、1、2、および10に更新されます。length プロパティに既存の長さよりも短い値を代入した場合、配列は切り詰められます。

```
var my_array:Array = new Array();  
trace(my_array.length); // initial length is 0  
my_array[0] = "a";  
trace(my_array.length); // my_array.length is updated to 1  
my_array[1] = "b";  
trace(my_array.length); // my_array.length is updated to 2  
my_array[9] = "c";  
trace(my_array.length); // my_array.length is updated to 10  
trace(my_array);  
// displays:  
// a,b,undefined,undefined,undefined,undefined,undefined,c  
  
// if the length property is now set to 5, the array will be truncated  
my_array.length = 5;  
trace(my_array.length); // my_array.length is updated to 5  
trace(my_array); // outputs: a,b,undefined,undefined,undefined
```

NUMERIC (Array.NUMERIC プロパティ)

public static NUMERIC : Number

ストリングベースのソートではなく、数値によるソートを表します。ストリングベースのソートは、デフォルトの設定であり、ソートするときに数値をストリングとして処理します。たとえば、ストリングベースのソートでは、10 は 3 よりも前に配置されます。数値によるソートではエレメントは数値として処理されるので、3 は 10 よりも前に配置されます。この定数は、sort() メソッドまたは sortOn() メソッドの options パラメータで使用できます。この定数の値は 16 です。

関連項目

[sort \(Array.sort メソッド\)](#), [sortOn \(Array.sortOn メソッド\)](#)

pop (Array.pop メソッド)

public pop() : Object

配列の最後のエレメントを削除して、そのエレメントの値を返します。

戻り値

[Object](#) - 指定された配列の最後のエレメントの値。

例

次のコードでは、エレメントが 4 つある配列 myPets_array を作成した後、その最後のエレメントを削除します。

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var popped:Object = myPets_array.pop();
trace(popped); // Displays fish.
trace(myPets_array); // Displays cat,dog,bird.
```

関連項目

[push \(Array.push メソッド\)](#), [shift \(Array.shift メソッド\)](#), [unshift \(Array.unshift メソッド\)](#)

push (Array.push メソッド)

public push(value:Object) : Number

エレメントを配列の最後に追加して、配列の新しい長さを返します。

パラメータ

`value:Object` - 配列に付加される値。

戻り値

`Number` - 新しい配列の長さを表す整数。

例

次の例では、`cat` と `dog` の 2 つのエレメントを持つ配列 `myPets_array` を作成します。2 行目で配列に 2 つのエレメントを追加します。

`push()` メソッドから配列の新しい長さが返されるので、最終行の `trace()` ステートメントは `myPets_array` の新しい長さ (4) を [出力] パネルに表示します。

```
var myPets_array:Array = new Array("cat", "dog");
var pushed:Number = myPets_array.push("bird", "fish");
trace(pushed); // Displays 4.
```

関連項目

[pop \(Array.pop メソッド\)](#), [shift \(Array.shift メソッド\)](#), [unshift \(Array.unshift メソッド\)](#)

RETURNINDEXEDARRAY

(Array.RETURNINDEXEDARRAY プロパティ)

public static RETURNINDEXEDARRAY : Number

`sort()` または `sortOn()` メソッドの呼び出し結果としてインデックス付き配列を返すオプションを表します。この定数は、`sort()` メソッドまたは `sortOn()` メソッドの `options` パラメータで使用できます。この定数を使用すると、ソートの結果を表す配列が返されますが、元の配列は変更されないままになるので、プレビュー機能やコピー機能になります。この定数の値は 8 です。

関連項目

[sort \(Array.sort メソッド\)](#), [sortOn \(Array.sortOn メソッド\)](#)

reverse (Array.reverse メソッド)

public reverse() : Void

配列の並びを反転させます。

例

次の例では、このメソッドを使用して、配列 numbers_array の並びを反転させます。

```
var numbers_array:Array = new Array(1, 2, 3, 4, 5, 6);
trace(numbers_array); // Displays 1,2,3,4,5,6.
numbers_array.reverse();
trace(numbers_array); // Displays 6,5,4,3,2,1.
```

shift (Array.shift メソッド)

public shift() : Object

配列の最初のエレメントを削除して、そのエレメントを返します。

戻り値

[Object](#) - 配列の最初のエレメント。

例

次のコードでは、配列 myPets_array を作成し、その配列から最初のエレメントを削除して変数 shifted に代入します。

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var shifted:Object = myPets_array.shift();
trace(shifted); // Displays "cat".
trace(myPets_array); // Displays dog,bird,fish.
```

関連項目

[pop \(Array.pop メソッド\)](#), [push \(Array.push メソッド\)](#), [unshift \(Array.unshift メソッド\)](#)

slice (Array.slice メソッド)

```
public slice([startIndex:Number], [endIndex:Number]) : Array
```

元の配列から一連のエレメントを取り出して、新しい配列を返します。元の配列は変更されません。

返される配列には、startIndex エレメントから endIndex エレメントまでのエレメント

(endIndex エレメント自体は除く) がすべて含まれます。

パラメータを何も渡さないと、元の配列の複製が作成されます。

パラメータ

`startIndex:Number` (オプション) - スライスの始点のインデックスを指定する数値。`start` が負の数値の場合、始点は配列の末尾から指定されます。つまり、`-1` が最後のエレメントです。

`endIndex:Number` (オプション) - スライスの終点のインデックスを指定する数値。このパラメータを省略すると、スライスには配列の始点から最後までのすべてのエレメントが取り込まれます。`end` が負の数値の場合、終点は配列の末尾から指定されます。つまり、`-1` が最後のエレメントです。

戻り値

`Array` - 元の配列から一連のエレメントを取り出した配列。

例

次の例では、まず 5 種類のペットで構成される配列を作成し、次に `slice()` を使用して、4 足動物のみを含む新しい配列を作成します。

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFourLeggedPets_array:Array = new Array();
var myFourLeggedPets_array = myPets_array.slice(0, 2);
trace(myFourLeggedPets_array); // Returns cat,dog.
trace(myPets_array); // Returns cat,dog,fish,canary,parrot.
```

次の例では、5 種類のペットで構成される配列を作成し、次に `slice()` に負の `start` パラメータを渡して、配列から最後の 2 つのエレメントをコピーします。

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFlyingPets_array:Array = myPets_array.slice(-2);
trace(myFlyingPets_array); // Traces canary,parrot.
```

次の例では、5 種類のペットで構成される配列を作成し、次に `slice()` に負の `end` パラメータを渡して、配列から中間のエレメントをコピーします。

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myAquaticPets_array:Array = myPets_array.slice(2,-2);
trace(myAquaticPets_array); // Returns fish.
```

sort (Array.sort メソッド)

public sort([compareFunction:Object], [options:Number]) : Array

配列内のエレメントをソートします。ソートは Unicode 値に基づいて実行されます (ASCII は Unicode のサブセットです)。

Array.sort() のデフォルトの動作は、次のようにになります。

- ソートでは大文字と小文字が区別されます。Z が a よりも先になります。
- 昇順にソートされます。a が b よりも先になります。
- 配列はソート順を反映するように変更されます。同じソートフィールドを持つ複数のエレメントは、ソート済みの配列の中で連続的かつランダムに格納されます。
- 数値フィールドは、ストリングとしてソートされます。たとえば、"1" は "9" よりも小さいストリング値であるため、100 は 99 よりも先になります。

デフォルト設定と異なる設定を使用して配列のソートを行う場合は、options パラメータのエントリで説明されているソートオプションのいずれかを使用することも、ソート処理を行う独自のカスタム関数を作成することもできます。カスタム関数を作成する場合は、最初のパラメータ (compareFunction) としてカスタム関数の名前を使用して、sort() メソッドを呼び出すことで作成できます。

パラメータ

compareFunction:Object (オプション) - 配列内のエレメントのソート順を決定する比較関数。

エレメント A と B がある場合、compareFunction は次の 3 つの値のいずれかを返します。

- A が B の前に表示されるソート順の場合は -1
- A = B の場合は 0
- A が B の後に表示されるソート順の場合は 1

options:Number (オプション) - デフォルトのソート動作を変更する数値または定義済み定数の名前。複数指定する場合は、ビット単位の論理和 (OR) | 演算子で区切ります。options パラメータには次の値を指定できます。

- Array.CASEINSENSITIVE または 1
- Array.DESCENDING または 2
- Array.UNIQUESORT または 4
- Array.RETURNINDEXEDARRAY または 8
- Array.NUMERIC または 16

このパラメータの詳細については、「Array.sortOn() メソッド」を参照してください。

メモ: Array.sort() は ECMA-262 仕様を拡張した Flash 固有の機能です。ECMA-262 でも定義されていますが、配列ソートオプションは Flash Player 7 で実装したものです。

戻り値

[Array](#) - 戻り値は、次に示すように、指定されたパラメータによって異なります。

- options パラメータに値 4 または Array.UNIQUESORT を指定し、ソート対象の複数のエレメントにまったく同じソートフィールドがある場合は 0 が返されます。配列は変更されません。
- options パラメータに値 8 または Array.RETURNINDEXEDARRAY を指定すると、ソート結果を反映した配列が返されます。配列は変更されません。
- それ以外の場合、値は返されません。ソート順を反映するように配列が変更されます。

例

シンタックス1: 次に、options に値が渡された場合と渡されなかった場合の `Array.sort()` の例を示します。

```
var fruits_array:Array = new Array("oranges", "apples", "strawberries",
    "pineapples", "cherries");
trace(fruits_array); // Displays
    oranges,apples,strawberries,pineapples,cherries.
fruits_array.sort();
trace(fruits_array); // Displays
    apples,cherries,oranges,pineapples,strawberries.
trace(fruits_array); // Writes apples,cherries,oranges,pineapples,strawberries.
fruits_array.sort(Array.DESCENDING);
trace(fruits_array); // Displays
    strawberries,pineapples,oranges,cherries,apples.
trace(fruits_array); // Writes strawberries,pineapples,oranges,cherries,apples.
```

シンタックス2: 次に、`Array.sort()` と比較関数を組み合わせた例を示します。エントリは "名前: パスワード" でソートされます。エントリの名前の部分だけをソート条件として使用します。

```
var passwords_array:Array = new Array("mom:glam", "ana:ring", "jay:mag",
    "anne:home", "regina:silly");
function order(a, b):Number {
    var name1:String = a.split(":")[0];
    var name2:String = b.split(":")[0];
    if (name1<name2) {
        return -1;
    } else if (name1>name2) {
        return 1;
    } else {
        return 0;
    }
}
trace("Unsorted:");
//Displays Unsorted:
trace(passwords_array);
//Displays mom:glam,ana:ring,jay:mag,anne:home,regina:silly.
//Writes mom:glam,ana:ring,jay:mag,anne:home,regina:silly
passwords_array.sort(order);
```

```
trace("Sorted:");
//Displays Sorted:
trace(passwords_array);
//Displays ana:ring,anne:home,jay:mag,mom:glam,regina:silly.
//Writes ana:ring,anne:home,jay:mag,mom:glam,regina:silly.
```

関連項目

| [ビット単位の論理和（OR）演算子](#), [sortOn \(Array.sortOn メソッド\)](#)

sortOn (Array.sortOn メソッド)

public sortOn(fieldName:[Object](#), [options:[Object](#)]) : Array

配列内のフィールド（複数のフィールドも可能）に基づいて、配列内のエレメントをソートします。配列は、次に示す特性を備えている必要があります。

- インデックス付き配列を対象とします。連想配列は対象外です。
- 配列の各エレメントは、プロパティがあるオブジェクトを保持するものとします。
- すべてのオブジェクトには共通のプロパティが少なくとも 1 つあるものとします。このようなプロパティをフィールドと言います。

fieldName パラメータを複数指定する場合、先頭のフィールドが第 1 ソートフィールド、2 番目のフィールドが第 2 ソートフィールド、（以下同様）と見なされます。ソートは Unicode 値に基づいて実行されます（ASCII は Unicode のサブセットです）。fieldName パラメータで指定されたフィールドが、比較対象のいずれのエレメントにも含まれていない場合、そのフィールドは undefined と見なされます。ソート済みの配列では、エレメントが連続的かつランダムに格納されます。

Array.sortOn() のデフォルトの動作は、次のようになります。

- ソートでは大文字と小文字が区別されます。Z が a よりも先になります。
- 昇順にソートされます。a が b よりも先になります。
- 配列はソート順を反映するように変更されます。同じソートフィールドを持つ複数のエレメントは、ソート済みの配列の中で連続的かつランダムに格納されます。
- 数値フィールドは、ストリングとしてソートされます。たとえば、"1" は "9" よりも小さいストリング値であるため、100 は 99 よりも先になります。

options パラメータを使用して、デフォルトのソート動作を上書きすることができます。単純な配列（たとえば、1 つのフィールドだけを持つ配列）をソートする場合、または options パラメータでサポートされていないソート順序を指定する場合は、Array.sort() を使用します。

複数のフラグを渡すには、ビット単位の論理和（OR）（|）演算子で区切ります。

```
my_array.sortOn(someFieldName, Array.DESCENDING | Array.NUMERIC);
```

パラメータ

fieldName: Object - ソート値として使用するフィールドを指定するストリング、または、先頭のエレメントが第1ソートフィールド、2番目が第2ソートフィールド、(以下同様)を表す配列。

options: Object (オプション) - ソート動作を変更する数値または定義済み定数の名前。複数指定する場合は、ビット単位の論理和(OR) | 演算子で区切れます。options パラメータには次の値を指定できます。

- Array.CASEINSENSITIVE または 1
- Array.DESCENDING または 2
- Array.UNIQUESORT または 4
- Array.RETURNINDEXEDARRAY または 8
- Array.NUMERIC または 16

数値形式(2)ではなく、ストリング形式のフラグ(DESCENDINGなど)を使用すると、コードヒントが有効になります。

戻り値

Array - 戻り値は、次に示すように、指定されたパラメータによって異なります。

- options パラメータに値 4 または Array.UNIQUESORT を指定し、ソート対象の複数のエレメントにまったく同じソートフィールドがある場合は 0 が返されます。配列は変更されません。
- options パラメータに値 8 または Array.RETURNINDEXEDARRAY を指定すると、ソート結果を反映した配列が返されます。配列は変更されません。
- それ以外の場合、値は返されません。ソート順を反映するように配列が変更されます。

例

次の例では、新しい配列を作成し、その配列を name フィールドと city フィールドに基づいてソートします。最初の例では、第1ソート値に name を、第2ソート値に city を使用します。2番目の例では、第1ソート値に city を、第2ソート値に name を使用します。

```
var rec_array:Array = new Array();
rec_array.push({name: "john", city: "omaha", zip: 68144});
rec_array.push({name: "john", city: "kansas city", zip: 72345});
rec_array.push({name: "bob", city: "omaha", zip: 94010});
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn(["name", "city"]);
```

```
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn(["city", "name"]);
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, kansas city
// bob, omaha
// john, omaha
```

次の例では、オブジェクト配列を使用して、options パラメータの使用法を示します。

```
var my_array:Array = new Array();
my_array.push({password: "Bob", age:29});
my_array.push({password: "abcd", age:3});
my_array.push({password: "barb", age:35});
my_array.push({password: "catchy", age:4});

password フィールドに対してデフォルトのソートを実行すると、次の結果が生成されます。

my_array.sortOn("password");
// Bob
// abcd
// barb
// catchy

password フィールドに対して大文字と小文字を区別しないソートを実行すると、次の結果が生成されます。

my_array.sortOn("password", Array.CASEINSENSITIVE);
// abcd
// barb
// Bob
// catchy

password フィールドに対して、大文字と小文字を区別しない降順ソートを実行すると、次の結果が生成されます。

my_array.sortOn("password", Array.CASEINSENSITIVE | Array.DESCENDING);
// catchy
// Bob
// barb
// abcd
```

`age` フィールドに対してデフォルトのソートを実行すると、次の結果が生成されます。

```
my_array.sortOn("age");
// 29
// 3
// 35
// 4
```

`age` フィールドに対して数値ソートを実行すると、次の結果が生成されます。

```
my_array.sortOn("age", Array.NUMERIC);
// my_array[0].age = 3
// my_array[1].age = 4
// my_array[2].age = 29
// my_array[3].age = 35
```

`age` フィールドに対して数値の降順ソートを実行すると、次の結果が生成されます。

```
my_array.sortOn("age", Array.DESCENDING | Array.NUMERIC);
// my_array[0].age = 35
// my_array[1].age = 29
// my_array[2].age = 4
// my_array[3].age = 3
```

`Array.RETURNEDINDEXARRAY` ソートオプションを使用する場合は、別の配列に戻り値を代入する必要があります。元の配列は変更されません。

```
var indexArray:Array = my_array.sortOn("age", Array.RETURNINDEXEDARRAY);
```

関連項目

| [ビット単位の論理和（OR）演算子 , sort \(Array.sort メソッド\)](#)

splice (Array.splice × ソッド)

```
public splice(startIndex:Number, [deleteCount:Number], [value:Object]):Array
```

配列のエレメントを追加および削除します。このメソッドは、コピーを作成しないで、配列を変更します。

パラメータ

`startIndex:Number` - 挿入または削除を開始する配列エレメントのインデックスを示す整数。負の整数を指定すると、配列の末尾を基準として位置を指定できます（たとえば、`-1` は配列の最後のエレメントです）。

`deleteCount:Number` (オプション) - 削除するエレメント数を示す整数。この数には、`startIndex` パラメータで指定するエレメントが含まれます。`deleteCount` パラメータに値を指定しないと、`startIndex` パラメータで指定した配列エレメントから最後の配列エレメントまでの値がすべて削除されます。値として `0` を指定すると、エレメントは削除されません。

`value:Object` (オプション) - `startIndex` パラメータで指定した挿入箇所に挿入する値を指定します。

戻り値

[Array](#) - 元の配列から削除したエレメントが含まれる配列。

例

次の例では、配列を1つ作成し、spliceメソッドのstartIndexパラメータにエレメントインデックス1を使用してスプライスします。これにより、2番目以降のすべてのエレメントが配列から削除され、元の配列にはインデックス0だけが残ります。

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
trace( myPets_array.splice(1) ); // Displays dog,bird,fish.
trace( myPets_array ); // cat
```

次の例では、配列を1つ作成し、spliceメソッドのstartIndexパラメータにエレメントインデックス1を、deleteCountパラメータに数値2を使用してスプライスします。これにより、2番目以降の2つのエレメントが配列から削除され、元の配列には最初と最後のエレメントだけが残ります。

```
var myFlowers_array:Array = new Array("roses", "tulips", "lilies", "orchids");
trace( myFlowers_array.splice(1,2) ); // Displays tulips,lilies.
trace( myFlowers_array ); // roses,orchids
```

次の例では、配列を1つ作成し、spliceメソッドのstartIndexパラメータにエレメントインデックス1を、deleteCountパラメータに数値0を、valueパラメータにストリングchairを使用してスプライスします。この場合、元の配列からは何も削除されず、ストリングchairがインデックス1に追加されます。

```
var myFurniture_array:Array = new Array("couch", "bed", "desk", "lamp");
trace( myFurniture_array.splice(1,0, "chair" ) ); // Displays empty array.
trace( myFurniture_array ); // displays couch,chair,bed,desk,lamp
```

toString (Array.toString メソッド)

public `toString()` : String

指定されたArrayオブジェクト内のエレメントを表すストリングを返します。インデックス0から最大インデックスまでの配列内のすべてのエレメントを、カンマで区切られた連結ストリングに変換して返します。カスタムセパレータを指定するには、`Array.join()`メソッドを使用します。

戻り値

[String](#) - ストリング。

例

次の例では、my_array を作成し、それをストリングに変換します。

```
var my_array:Array = new Array();
my_array[0] = 1;
my_array[1] = 2;
my_array[2] = 3;
my_array[3] = 4;
my_array[4] = 5;
trace(my_array.toString()); // Displays 1,2,3,4,5.
```

上記の例では、trace ステートメントの結果として、1,2,3,4,5 が出力されます。

関連項目

[split \(String.split メソッド\)](#), [join \(Array.join メソッド\)](#)

UNIQUESORT (Array.UNIQUESORT プロパティ)

public static UNIQUESORT : Number

一意性ソート要件を表します。この定数は、sort() メソッドまたは sortOn() メソッドの options パラメータで使用できます。一意性ソートオプションを指定すると、ソート対象に含まれる任意の 2 つのエレメントまたはフィールドの値が同じである場合にソートが中止されます。この定数の値は 4 です。

関連項目

[sort \(Array.sort メソッド\)](#), [sortOn \(Array.sortOn メソッド\)](#)

unshift (Array.unshift メソッド)

public unshift(value:Object) : Number

エレメントを配列の先頭に追加して、配列の新しい長さを返します。

パラメータ

value:Object - 配列の先頭に挿入される数値、エレメント、または変数。

戻り値

Number - 新しい配列の長さを表す整数。

例

次の例では、`Array.unshift()` メソッドの使用法を示します。

```
var pets_array:Array = new Array("dog", "cat", "fish");
trace( pets_array ); // Displays dog,cat,fish.
pets_array.unshift("ferrets", "gophers", "engineers");
trace( pets_array ); // Displays ferrets,gophers,engineers,dog,cat,fish.
```

関連項目

[pop \(Array.pop メソッド\)](#), [push \(Array.push メソッド\)](#), [shift \(Array.shift メソッド\)](#)

Boolean

`Object`

|
+-`Boolean`

```
public class Boolean
extends Object
```

`Boolean` クラスは、標準 JavaScript の `Boolean` オブジェクトと同じ機能を持つラッパーオブジェクトです。`Boolean` クラスを使用して、`Boolean` オブジェクトのプリミティブなデータ型またはストリング表現を調べることができます。

`Boolean` オブジェクトのメソッドを呼び出すには、コンストラクタの新しい `Boolean()` を使用してこのオブジェクトを作成する必要があります。

プロパティ一覧

`Object` クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve
プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>Boolean([value:Object])</code>	<code>Boolean</code> オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>toString() : String</code>	Boolean オブジェクトのストリング表現("true" または "false")を返します。
	<code>valueOf() : Boolean</code>	指定された Boolean オブジェクトのプリミティブな値のタイプが true の場合は true を、それ以外の場合は false を返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

Boolean コンストラクタ

`public Boolean([value:Object])`

Boolean オブジェクトを作成します。value パラメータを省略すると、Boolean オブジェクトは値 false で初期化されます。value パラメータの値を指定すると、メソッドによって評価され、評価結果はグローバル Boolean() 関数の規則に従ってブール値として返されます。

パラメータ

`value:Object` (オプション) - 任意の式。デフォルト値は false です。

例

次のコードでは、新しい空の Boolean オブジェクトとして myBoolean を作成します。

```
var myBoolean:Boolean = new Boolean();
```

toString (Boolean.toString メソッド)

`public toString() : String`

Boolean オブジェクトのストリング表現("true" または "false")を返します。

戻り値

`String` - ストリング。"true" または "false"。

例

次の例では、Boolean 型の変数を作成し、`toString()` を使って値を `trace` ステートメントで使用するストリングに変換します。

```
var myBool:Boolean = true;
trace("The value of the Boolean myBool is: " + myBool.toString());
myBool = false;
trace("The value of the Boolean myBool is: " + myBool.toString());
```

valueOf (Boolean.valueOf メソッド)

`public valueOf(): Boolean`

指定された Boolean オブジェクトのプリミティブな値のタイプが `true` の場合は `true` を、それ以外の場合は `false` を返します。

戻り値

`Boolean` - ブール値。

例

次の例では、このメソッドがどのように動作するか、および新しい Boolean オブジェクトのプリミティブな値のタイプが `false` であることを示します。

```
var x:Boolean = new Boolean();
trace(x.valueOf()); // false
x = (6==3+3);
trace(x.valueOf()); // true
```

Button

```
Object
|
+-Button
```

```
public class Button
extends Object
```

SWF ファイルのすべてのボタンシンボルは、`Button` オブジェクトのインスタンスです。プロパティインスペクタを使用して、ボタンにインスタンス名を付けることができます。また `Button` クラスのメソッドとプロパティを使用して、ActionScript でボタンを操作できます。ボタンのインスタンス名は、ムービーエクスプローラに表示されます。[アクション] パネルの [ターゲットパスの挿入] ダイアログボックスにも表示されます。

`Button` クラスは `Object` クラスから継承します。

関連項目

[Object](#)

プロパティ一覧

オプション	プロパティ	説明
	<code>_alpha:Number</code>	<code>my_btn</code> で指定されているボタンのアルファ透明度の値です。
	<code>enabled:Boolean</code>	ボタンが有効であるか無効であるかを指定するブール値です。
	<code>_focusrect:Boolean</code>	入力フォーカスがあるボタンの周囲に黄色の矩形を表示するかどうかを指定するブール値です。
	<code>_height:Number</code>	ボタンの高さ(ピクセル単位)です。
	<code>_highquality:Number</code>	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに <code>Button._quality</code> を使用します。現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。
	<code>_name:String</code>	<code>my_btn</code> で指定されているボタンのインスタンス名です。
	<code>_parent:MovieClip</code>	現在のボタンを含むムービークリップを指す参照です。
	<code>_quality:String</code>	プロパティ(グローバル)。SWF ファイルに使用するレンダリング品質を設定または取得します。
	<code>_rotation:Number</code>	ボタンの元の位置からの回転角(度単位)です。
	<code>_soundbuftime:Number</code>	サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。
	<code>tabEnabled:Boolean</code>	<code>my_btn</code> が自動タブ順に含まれているかどうかを指定します。
	<code>tabIndex:Number</code>	SWF ファイル内のオブジェクトのタブ順をカスタマイズできます。
	<code>_target:String (読み取り専用)</code>	<code>my_btn</code> で指定されているボタンインスタンスのターゲットパスを返します。
	<code>trackAsMenu:Boolean</code>	他のボタンまたはムービークリップがマウスまたはスタイルスから解放イベントを受け取ることができるかどうかを示すブール値です。
	<code>_url:String (読み取り専用)</code>	ボタンの配置された SWF ファイルの URL を取得します。
	<code>_visible:Boolean</code>	<code>my_btn</code> で指定されているボタンを表示するかどうかを示すブール値です。
	<code>_width:Number</code>	ボタンの幅(ピクセル単位)です。

オプション	プロパティ	説明
	<code>_x:Number</code>	親ムービークリップのローカル座標を基準にしたボタンの x 座標を設定する整数です。
	<code>_xmouse:Number</code> (読み取り専用)	<code>hasMouse</code> が <code>true</code> の場合にボタンを基準にしたマウス位置の x 座標を返します。
	<code>_xscale:Number</code>	ボタンの基準点から適用した場合のボタンの水平方向の拡大 / 縮小率(パーセント単位)です。
	<code>_y:Number</code>	親ムービークリップのローカル座標を基準にしたボタンの y 座標です。
	<code>_ymouse:Number</code> (読み取り専用)	ボタンを基準にしたマウス位置の y 座標を返します。
	<code>_yscale:Number</code>	ボタンの基準点から適用した場合のボタンの垂直方向の拡大 / 縮小率(パーセント単位)です。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

イベント一覧

イベント	説明
<code>onDragOut</code> = <code>function() {}</code>	ボタン上でマウスボタンを押し、一度ボタンの外側に移動すると、呼び出されます。
<code>onDragOver</code> = <code>function() {}</code>	ボタンの上でマウスボタンを押し、一度ボタンの外にドラッグして移動してからドラッグしてボタン上に戻ると、呼び出されます。
<code>onKeyDown</code> = <code>function() {}</code>	ボタンにキーボードフォーカスがあるときにキーを押すと、呼び出されます。
<code>onKeyUp</code> = <code>function() {}</code>	ボタンに入力フォーカスがあるときにキーを離すと、呼び出されます。
<code>onKillFocus</code> = <code>function(newFocus: Object) {}</code>	ボタンがキーボードフォーカスを失うと、呼び出されます。
<code>onPress</code> = <code>function() {}</code>	ボタンを押すと呼び出されます。
<code>onRelease</code> = <code>function() {}</code>	ボタンを離すと呼び出されます。

イベント	説明
<code>onReleaseOutside = function() {}</code>	ポインタがボタン内にあるときにマウスボタンを押した後で、ポインタがボタン外にあるときにマウスを離すと、呼び出されます。
<code>onRollOut = function() {}</code>	ボタンがフォーカスを失うと、呼び出されます。
<code>onRollOver = function() {}</code>	ボタンがフォーカスを受け取ると、呼び出されます。
<code>onSetFocus = function(oldFocus: Object) {}</code>	ボタンがキーボードフォーカスを受け取ると、呼び出されます。

メソッド一覧

オプション	シグネチャ	説明
	<code>getDepth() : Number</code>	ボタンインスタンスの深度を返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

_alpha (Button._alpha プロパティ)

`public _alpha : Number`

my_btn で指定されているボタンのアルファ透明度の値です。有効な値は 0 (完全な透明) ~ 100 (完全な不透明) です。デフォルト値は 100 です。`_alpha` が 0 に設定されているボタン内のオブジェクトは表示されませんが、その状態でもアクティブです。

例

次のコードは、ユーザーがボタンをクリックすると、ボタン myBtn_btn の `_alpha` プロパティを 50% に設定します。まず、Button インスタンスをステージに追加します。次に、インスタンスに `myBtn_btn` という名前を付けます。最後に、フレーム 1 を選択し、次のコードを [アクション] パネル内に置きます。

```
myBtn_btn.onRelease = function(){  
    this._alpha = 50;  
};
```

関連項目

[_alpha \(MovieClip._alpha プロパティ\)](#), [_alpha \(TextField._alpha プロパティ\)](#)

enabled (Button.enabled プロパティ)

public enabled : Boolean

ボタンが有効であるか無効であるかを指定する布尔値です。ボタンが無効になっているとき (enabled プロパティが false に設定されているとき)、ボタンは表示されますが、クリックできません。デフォルト値は true です。このプロパティは、ナビゲーションの一部を無効にする場合に便利です。たとえば、現在表示されているページのボタンを無効にしてクリックできないようにすることで、ページのリロードを禁止できます。

例

次の例では、ボタンのクリックを無効および有効にする方法を示します。myBtn1_btn と myBtn2_btn の 2 つのボタンがステージ上にあります。次の ActionScript を追加して、myBtn2_btn ボタンをクリックできないようにします。最初に、2 つのボタンインスタンスをステージに追加します。次に、それぞれのインスタンスに myBtn1_btn および myBtn2_btn という名前を付けます。最後に、フレーム 1 に次のコードを追加して、ボタンのクリックを有効または無効にします。

```
myBtn1_btn.enabled = true;
myBtn2_btn.enabled = false;

//button code
// the following function will not get called
// because myBtn2_btn.enabled was set to false
myBtn1_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};

myBtn2_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};
```

_focusrect (Button._focusrect プロパティ)

public _focusrect : Boolean

入力フォーカスがあるボタンの周囲に黄色の矩形を表示するかどうかを指定する布尔値です。このプロパティは、グローバル _focusrect プロパティの設定を上書きできます。デフォルトでは、ボタンインスタンスの _focusrect プロパティは null です。つまり、ボタンインスタンスはグローバル _focusrect プロパティを上書きしません。あるボタンインスタンスの _focusrect プロパティを true または false に設定した場合、そのボタンインスタンスのグローバル _focusrect プロパティの設定は上書きされます。

Flash Player 4 または Flash Player 5 の SWF ファイルでは、_focusrect プロパティがグローバル _focusrect プロパティを制御します。これはブール値です。Flash Player 6 以降では、この動作は、個々のムービークリップで _focusrect プロパティをカスタマイズできるように変更されています。

_focusrect プロパティが `false` である場合、そのボタンのキーボードナビゲーションは Tab キーに限定されます。Enter キーや矢印キーを含め、他のキーはすべて無視されます。すべてのキーナビゲーションを元に戻すには、_focusrect を `true` に設定する必要があります。

メモ: Flash Lite 2.0 プレーヤーでは、_focusrect プロパティが無効になっている場合（つまり、`Button.focusRect` が `false` である場合）、ボタンはすべてのイベントを受け取ります。この動作は、Flash Player の動作とは異なります。Flash Player では、_focusrect プロパティが無効である場合、ボタンは `rollOver` および `rollOut` イベントを受け取ますが、`press` および `release` イベントは受け取れません。

また、Flash Lite 2.0 では、`fscommand2 SetFocusRectColor` コマンドを使用して、フォーカス矩形のカラーを変更できます。この動作も Flash Player とは異なります。Flash Player では、フォーカス矩形のカラーは黄色に限定されています。

例

次の例では、ブラウザウィンドウでフォーカスを受け取ったときに、SWF ファイル内の指定されたボタンインスタンスの周囲の黄色の矩形を非表示にします。`myBtn1_btn`、`myBtn2_btn`、`myBtn3_btn` の 3 つのボタンを作成し、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
myBtn2_btn._focusrect = false;
```

テスト環境で SWF ファイルをテストする場合は、[制御]-[キーボードショートカットを無効] を選択してキーボードショートカットを無効にしてください。

`_focusrect` が無効になっている場合は、Enter キーまたはスペースキーを押してもこのボタンのコードを実行できません。

getDepth (Button.getDepth メソッド)

```
public getDepth():Number
```

ボタンインスタンスの深度を返します。

各ムービークリップ、ボタン、およびテキストフィールドには関連付けられた一意の深度があり、どのように他のオブジェクトの手前や背後に表示されるかが決まります。深度の高い方のオブジェクトが手前に表示されます。

戻り値

`Number` - ボタンインスタンスの深度。

例

myBtn1_btn と myBtn2_btn をステージに作成した場合、次の ActionScript を使用して深度をトレースできます。

```
trace(myBtn1_btn.getDepth());
trace(myBtn2_btn.getDepth());
```

"buttonMovie.swf" という名前の SWF ファイルをこのドキュメントにロードする場合、メイン SWF ファイルにある別のボタンを使用して、その SWF ファイル内のボタン myBtn4_btn の深度をトレースできます。

```
this.createEmptyMovieClip("myClip_mc", 999);
myClip_mc.loadMovie("buttonMovie.swf");
myBtn3_btn.onRelease = function(){
    trace(myClip_mc.myBtn4_btn.getDepth());
};
```

これらのボタンのうち、メイン SWF ファイル内のボタンとロードされた SWF ファイル内のボタンの深度は同じです。このように紛らわしいのは、"buttonMovie.swf" が深度 999 でロードされたため、そこに含まれるボタンの深度も、メイン SWF ファイル内のボタンを基準にした 999 になるためです。それぞれのムービークリップは独自の内部 z 順序を持つこと、つまり、それぞれのムービークリップには独自の深度値のセットがあることに注意してください。2つのボタンは同じ深度値を持つことができますが、その値は、同じ z 順序内の他のオブジェクトとの関連においてのみ意味を持ちます。このケースでは、ボタンは同じ深度値を持ちますが、値はそれ異なるムービークリップに関連付けられています。つまり、メイン SWF ファイル内のボタンの深度値はメインタイムラインの z 順序に関連付けられており、ロードされた SWF ファイル内のボタンの深度値はムービークリップ myClip_mc の内部 z 順序に関連付けられています。

関連項目

[getDepth \(MovieClip.getDepth メソッド\)](#), [getDepth \(TextField.getDepth メソッド\)](#),
[getInstanceAtDepth \(MovieClip.getInstanceAtDepth メソッド\)](#)

_height (Button._height プロパティ)

public _height : Number

ボタンの高さ(ピクセル単位)です。

例

次の例では、my_btn という名前のボタンの高さと幅を、特定の高さと幅に設定します。

```
my_btn._width = 500;
my_btn._height = 200;
```

_highquality (Button._highquality プロパティ)

public _highquality : Number

非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに Button._quality を使用します。

現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。ビットマップスミングを常にオンにして最高品質を適用するには、2(最高品質)を指定します。アンチエイリアス処理を適用するには、1(高品質)を指定します。SWF ファイルにアニメーションが含まれない場合は、ビットマップは滑らかになります。これはデフォルト値です。アンチエイリアス処理を避けるには、0(低品質)を指定します。

例

ボタンインスタンスをステージに追加し、myBtn_btn という名前を付けます。楕円ツールを使用して、線と塗りのカラーを持つ楕円をステージ上に描きます。フレーム 1 を選択し、[アクション] パネルを使用して次の ActionScript を追加します。

```
myBtn_btn.onRelease = function() {  
    myBtn_btn._highquality = 0;  
};
```

myBtn_btn をクリックすると、円の線がギザギザになります。次の ActionScript を追加して、この効果を SWF にグローバルに適用することもできます。

```
_quality = 0;
```

関連項目

[_quality \(Button._quality プロパティ\)](#), [_quality プロパティ](#)

_name (Button._name プロパティ)

public _name : String

my_btn で指定されているボタンのインスタンス名です。

例

次の例では、SWF ファイルの現在のタイムライン内に存在する Button インスタンスのインスタンス名をすべてトレースします。

```
for (i in this) {  
    if (this[i] instanceof Button) {  
        trace(this[i]._name);  
    }  
}
```

onDragOut (Button.onDragOut ハンドラ)

```
onDragOut = function() {}
```

ボタン上でマウスボタンを押し、一度ボタンの外側に移動すると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

メモ : Flash Lite 2.0 では、onDragOut イベントハンドラは、System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、ポインタをボタンの外にドラッグしたときにステートメントを実行する方法を示します。
my_btn という名前のボタンをステージ上に作成し、タイムラインのフレームに次の ActionScript を追加します。

```
my_btn.onDragOut = function() {
    trace("onDragOut: "+this._name);
};

my_btn.onDragOver = function() {
    trace("onDragOver: "+this._name);
};
```

onDragOver (Button.onDragOver ハンドラ)

```
onDragOver = function() {}
```

ボタンの上でマウスボタンを押し、一度ボタンの外にドラッグして移動してからドラッグしてボタンに戻ると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

メモ : Flash Lite 2.0 では、onDragOver イベントハンドラは、System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、[出力] パネルに trace() ステートメントを送る onDragOver ハンドラの関数を定義します。my_btn という名前のボタンをステージ上に作成し、タイムラインに次の ActionScript を追加します。

```
my_btn.onDragOut = function() {
    trace("onDragOut: "+this._name);
};

my_btn.onDragOver = function() {
    trace("onDragOver: "+this._name);
};
```

SWF ファイルをテストするときは、ポインタをいったんボタンインスタンスの外にドラッグします。次に、マウスボタンを押したまま、再びポインタをボタンインスタンス上にドラッグします。[出力] パネルにその操作が記録されます。

関連項目

[onDragOut \(Button.onDragOut ハンドラ \)](#)

onKeyDown (Button.onKeyDown ハンドラ)

```
onKeyDown = function() {}
```

ボタンにキーボードフォーカスがあるときにキーを押すと、呼び出されます。onKeyDown イベントハンドラにはパラメータは渡されません。Key.getAscii() メソッドと Key.getCode() メソッドを使用して、どのキーが押されたかを調べることができます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

例

次の例では、onKeyDown ハンドラが呼び出されたときに [出力] パネルにテキストを送る関数を定義します。my_btn という名前のボタンをステージ上に作成し、タイムラインのフレームに次の ActionScript を追加します。

```
my_btn.onKeyDown = function() {
    trace("onKeyDown: "+this._name+ " (Key: "+getKeyPressed()+" )");
};

function getKeyPressed():String {
    var theKey:String;
    switch (Key.getAscii()) {
        case Key.BACKSPACE :
            theKey = "BACKSPACE";
            break;
        case Key.SPACE :
            theKey = "SPACE";
            break;
        default :
            theKey = chr(Key.getAscii());
    }
    return theKey;
}
```

[制御]-[ムービープレビュー] を選択して SWF ファイルをテストします。テスト環境では、[制御]-[キーボードショートカットを無効] を選択してください。次に、Tab キーを押してボタンにフォーカスを移動し (my_btn インスタンスの周囲に黄色の矩形が表示されます)、キーボードのキーを押してテキストを入力します。キーを押して入力したテキストが [出力] パネルに表示されます。

関連項目

[onKeyUp \(Button.onKeyUp ハンドラ \), getAscii \(Key.getAscii メソッド \), getCode \(Key.getCode メソッド \)](#)

onKeyUp (Button.onKeyUp ハンドラ)

```
onKeyUp = function() {}
```

ボタンに入力フォーカスがあるときにキーを離すと、呼び出されます。onKeyUp イベントハンドラにはパラメータは渡されません。Key.getAscii() メソッドと Key.getCode() メソッドを使用して、どのキーが押されたか調べることができます。

例

次の例では、onKeyDown ハンドラが呼び出されたときに [出力] パネルにテキストを送る関数を定義します。my_btn という名前のボタンをステージ上に作成し、タイムラインのフレームに次の ActionScript を追加します。

```
my_btn.onKeyDown = function() {
    trace("onKeyDown: "+this._name+ " (Key: "+getKeyPressed()+" )");
};

my_btn.onKeyUp = function() {
    trace("onKeyUp: "+this._name+ " (Key: "+getKeyPressed()+" )");
};

function getKeyPressed():String {
    var theKey:String;
    switch (Key.getAscii()) {
        case Key.BACKSPACE :
            theKey = "BACKSPACE";
            break;
        case Key.SPACE :
            theKey = "SPACE";
            break;
        default :
            theKey = chr(Key.getAscii());
    }
    return theKey;
}
```

Ctrl + Enter を押して SWF ファイルをテストします。テスト環境では、[制御]-[キーボードショートカットを無効] を選択してください。次に、Tab キーを押してボタンにフォーカスを移動し (my_btn インスタンスの周囲に黄色の矩形が表示されます) 、キーボードのキーを押してテキストを入力します。キーを押して入力したテキストが [出力] パネルに表示されます。

関連項目

[onKeyDown \(Button.onKeyDown ハンドラ \)](#), [getAscii \(Key.getAscii メソッド\)](#), [getCode \(Key.getCode メソッド\)](#)

onKillFocus (Button.onKillFocus ハンドラ)

```
onKillFocus = function(newFocus:Object) {}
```

ボタンがキーボードフォーカスを失うと、呼び出されます。onKillFocus ハンドラは、1つのパラメータ newFocus を受け取ります。このパラメータは、フォーカスを受け取る新しいオブジェクトを表すオブジェクトです。フォーカスを受け取るオブジェクトがない場合、newFocus の値は null です。

パラメータ

`newFocus:Object` - フォーカスを受け取るオブジェクト。

例

次の例では、ボタンがフォーカスを失ったときにステートメントを実行する方法を示します。my_btn という名前のボタンインスタンスをステージ上に作成し、タイムラインのフレーム1に次の ActionScript を追加します。

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
output_txt.wordWrap = true;
output_txt.multiline = true;
output_txt.border = true;
my_btn.onKillFocus = function() {
    output_txt.text = "onKillFocus: "+this._name+newline+output_txt.text;
};
```

SWF ファイルをブラウザウィンドウでテストするために、Tab キーを使用してウィンドウ内のエレメント間でフォーカスを移動します。ボタンインスタンスがフォーカスを失うと、テキストが output_txt テキストフィールドに送られます。

onPress (Button.onPress ハンドラ)

```
onPress = function() {}
```

ボタンを押すと呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

例

次の例では、onPress ハンドラが呼び出されたときに trace() ステートメントを [出力] パネルに送る関数を定義します。

```
my_btn.onPress = function () {
    trace ("onPress called");
};
```

onRelease (Button.onRelease ハンドラ)

```
onRelease = function() {}
```

ボタンを離すと呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

例

次の例では、onRelease ハンドラが呼び出されたときに trace() ステートメントを [出力] パネルに送る関数を定義します。

```
my_btn.onRelease = function () {
    trace ("onRelease called");
};
```

onReleaseOutside (Button.onReleaseOutside ハンドラ)

```
onReleaseOutside = function() {}
```

ポインタがボタン内にあるときにマウスボタンを押した後で、ポインタがボタン外にあるときにマウスを離すと、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

メモ : Flash Lite 2.0 では、onReleaseOutside イベントハンドラは、
System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、onReleaseOutside ハンドラが呼び出されたときに trace() ステートメントを [出力] パネルに送る関数を定義します。

```
my_btn.onReleaseOutside = function () {
    trace ("onReleaseOutside called");
};
```

onRollOut (Button.onRollOut ハンドラ)

```
onRollOut = function() {}
```

ボタンがフォーカスを失うと、呼び出されます。これは、ユーザーが現在選択しているボタン以外の別のボタンや領域をクリックしたときに発生します。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

例

次の例では、onRollOut ハンドラが呼び出されたときに trace() ステートメントを [出力] パネルに送る関数を定義します。

```
my_btn.onRollOut = function () {
    trace ("onRollOut called");
};
```

onRollOver (Button.onRollOver ハンドラ)

```
onRollOver = function() {}
```

ボタンがフォーカスを受け取ると、呼び出されます。ポインタがボタン領域の上に移動すると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。

例

次の例では、onRollOver ハンドラが呼び出されたときに trace() ステートメントを [出力] パネルに送る関数を定義します。

```
my_btn.onRollOver = function () {
    trace ("onRollOver called");
};
```

onSetFocus (Button.onSetFocus ハンドラ)

```
onSetFocus = function(oldFocus:Object) {}
```

ボタンがキーボードフォーカスを受け取ると、呼び出されます。oldFocus パラメータは、フォーカスを失うオブジェクトです。たとえば、Tab キーを押して入力フォーカスをテキストフィールドからボタンに移動すると、oldFocus にテキストフィールドのインスタンスが入ります。

前にフォーカスがあったオブジェクトが存在しない場合、oldFocus には null 値が入ります。

パラメータ

oldFocus:Object - キーボードフォーカスを失うオブジェクト。

例

次の例では、SWF ファイルのユーザーがあるボタンから別のボタンにフォーカスを移動したときにステートメントを実行する方法を示します。btn1_btn と btn2_btn の 2 つのボタンを作成し、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
Selection.setFocus(btn1_btn);
trace(Selection.getFocus());
btn2_btn.onSetFocus = function(oldFocus) {
    trace(oldFocus._name + " lost focus");
};
```

Ctrl + Enter を押して SWF ファイルをテストします。[制御]-[キーボードショートカットを無効] を選択していない場合は選択してください。まず、btn1_btn がフォーカスを得ます。次に、btn1_btn がフォーカスを失い、btn2_btn がフォーカスを得ます。情報が [出力] パネルに表示されます。

_parent (Button._parent プロパティ)

```
public _parent : MovieClip
```

現在のボタンを含むムービークリップを指す参照です。現在のオブジェクトとは、_parent を参照する ActionScript コードがあるオブジェクトです。

現在のムービークリップまたはオブジェクトの上位のムービークリップまたはオブジェクトへの相対パスを指定するには、_parent を使用します。_parent を使用して表示リストの複数のレベルを上に移動するには、次のようにします。

```
this._parent._parent._alpha = 20;
```

例

次の例で、ボタン my_btn はムービークリップ my_mc 内に配置されています。次のコードでは、_parent プロパティを使用してムービークリップ my_mc への参照を取得しています。

```
trace(my_mc.my_btn._parent);
[出力] パネルには次のように表示されます。
```

```
_level0.my_mc
```

関連項目

[_parent \(MovieClip._parent プロパティ\)](#), [_target \(MovieClip._target プロパティ\)](#), [_root プロパティ](#)

_quality (Button._quality プロパティ)

```
public _quality : String
```

プロパティ (グローバル)。SWF ファイルに使用するレンダリング品質を設定または取得します。デバイスフォントは常にエイリアス処理されるため、_quality プロパティには影響されません。

_quality プロパティは、次の値に設定できます。

- "LOW" 低いレンダリング品質。グラフィックスはアンチエイリアス処理されず、ピットマップはスムージングされません。
- "MEDIUM" 普通のレンダリング品質。グラフィックスは 2×2 ピクセルグリッドを使用してアンチエイリアス処理されますが、ピットマップはスムージングされません。これは、テキストを含まないムービーに適しています。
- "HIGH" 高いレンダリング品質。グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理されます。ピットマップは、ムービーが静的なものである場合は、スムージングされます。デフォルトのレンダリング品質設定です。

メモ : このプロパティを Button オブジェクトに対して指定することができますが、実際にはグローバルプロパティであるので、単に _quality という形で値を指定することもできます。

例

この例では、my_btn というボタンのレンダリング品質を LOW に設定します。

```
my_btn._quality = "LOW";
```

_rotation (Button._rotation プロパティ)

```
public _rotation : Number
```

ボタンの元の位置からの回転角 (度単位) です。時計回りに回転させる場合は 0 ~ 180 の値を指定します。反時計回りに回転させる場合は 0 ~ -180 の値を指定します。この範囲を超える値は、360 の倍数を加算または減算され、範囲内に収まる値になるように調整されます。たとえば、

```
my_btn._rotation = 450 というステートメントは my_btn._rotation = 90 と同義です。
```

例

次の例では、ステージ上の 2 つのボタンを回転させます。ステージ上に control_btn と my_btn の 2 つのボタンを作成します。my_btn は、回転している状態を判別できるように、完全な円にしないでください。次に、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var control_btn:Button;
var my_btn:Button;
control_btn.onRelease = function() {
    my_btn._rotation += 10;
};
```

ステージ上にもう1つのボタン myOther_btn を作成します。このボタンも、(回転している状態を判別できるように) 完全な円にしないでください。タイムラインのフレーム1に次の ActionScript を入力します。

```
var myOther_btn:Button;  
this.createEmptyMovieClip("rotater_mc", this.getNextHighestDepth());  
rotater_mc.onEnterFrame = function() {  
    myOther_btn._rotation += 2;  
};
```

関連項目

[_rotation \(MovieClip._rotation プロパティ\)](#), [_rotation \(TextField._rotation プロパティ\)](#)

_soundbuftime (Button._soundbuftime プロパティ)

public _soundbuftime : Number

サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。

メモ : このプロパティを Button オブジェクトに対して指定することができますが、実際はロードしたすべてのサウンドに適用されるグローバルプロパティであるので、単に _soundbuftime という形で値を指定することもできます。このプロパティを Button オブジェクトに対して設定すると、実際にグローバルプロパティに設定されます。

詳細および使用例については、_soundbuftime を参照してください。

関連項目

[_soundbuftime プロパティ](#)

tabEnabled (Button.tabEnabled プロパティ)

public tabEnabled : Boolean

my_btn が自動タブ順に含まれているかどうかを指定します。デフォルト値は undefined です。

tabEnabled プロパティが undefined または true である場合、オブジェクトは自動タブ順に含まれます。tabIndex プロパティにも値を設定すると、オブジェクトはカスタムタブ順にも含まれます。tabEnabled が false の場合は、TabIndex プロパティが設定されていても、オブジェクトは自動タブ順またはカスタムタブ順に含まれません。

例

次の ActionScript は、4つのボタンのいずれかの tabEnabled プロパティを false に設定するために使用されます。ただし、4つのボタン (one_btn、two_btn、three_btn、および four_btn) はすべて、tabIndex を使用してカスタムタブ順で配置されます。three_btn の tabIndex が設定されていても、そのインスタンスの tabEnabled が false に設定されているので、three_btn はカスタムタブ順にも自動タブ順にも含まれません。4つのボタンのタブ順を設定するには、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
three_btn.tabEnabled = false;  
two_btn.tabIndex = 1;  
four_btn.tabIndex = 2;  
three_btn.tabIndex = 3;  
one_btn.tabIndex = 4;
```

テスト環境で SWF ファイルをテストする場合は、[制御]-[キーボードショートカットを無効] を選択してキーボードショートカットを無効にしてください。

関連項目

[tabIndex \(Button.tabIndex プロパティ\)](#), [tabEnabled \(MovieClip.tabEnabled プロパティ\)](#), [tabEnabled \(TextField.tabEnabled プロパティ\)](#)

tabIndex (Button.tabIndex プロパティ)

public tabIndex : Number

SWF ファイル内のオブジェクトのタブ順をカスタマイズできます。ボタン、ムービークリップ、またはテキストフィールドインスタンスの tabIndex プロパティを設定できます。デフォルトの値は undefined です。

SWF ファイルに現在表示されているオブジェクトに tabIndex プロパティがある場合は、自動タブ順が無効になり、SWF ファイルのオブジェクトの tabIndex プロパティからタブ順が計算されます。カスタムタブ順には、tabIndex プロパティを持つオブジェクトのみが含まれます。

tabIndex プロパティは、通常、負以外の整数です。オブジェクトのタブ順は、その tabIndex プロパティに従って昇順に決定されます。tabIndex の値が 1 であるオブジェクトは、tabIndex の値が 2 であるオブジェクトよりも前になります。2つのオブジェクトの tabIndex が同じ値である場合、オブジェクトのタブ順は undefined になります。

tabIndex プロパティで定義されるカスタムタブ順は flat です。つまり、SWF ファイル内のオブジェクトの階層関係は無視されます。SWF ファイルで tabIndex プロパティを持つすべてのオブジェクトは、タブ順に従って配置されます。タブ順は tabIndex の値の順番に従います。tabIndex の値が同じである 2 つのオブジェクト間では、優先順が undefined になります。複数のオブジェクトの tabIndex に同じ値を使用しないでください。

例

次の ActionScript は、4つのボタンのいずれかの tabEnabled プロパティを false に設定するためを使用されます。ただし、4つのボタン (one_btn、two_btn、three_btn、および four_btn) はすべて、tabIndex を使用してカスタムタブ順で配置されます。three_btn の tabIndex が設定されていても、そのインスタンスの tabEnabled が false に設定されているので、three_btn はカスタムタブ順にも自動タブ順にも含まれません。4つのボタンのタブ順を設定するには、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
three_btn.tabEnabled = false;  
two_btn.tabIndex = 1;  
four_btn.tabIndex = 2;  
three_btn.tabIndex = 3;  
one_btn.tabIndex = 4;
```

テスト環境で SWF ファイルをテストする場合は、[制御]-[キーボードショートカットを無効] を選択してキーボードショートカットを無効にしてください。

関連項目

[tabEnabled \(Button.tabEnabled プロパティ\)](#), [tabChildren \(MovieClip.tabChildren プロパティ\)](#), [tabEnabled \(MovieClip.tabEnabled プロパティ\)](#), [tabIndex \(MovieClip.tabIndex プロパティ\)](#), [tabIndex \(TextField.tabIndex プロパティ\)](#)

_target (Button._target プロパティ)

public _target : [String](#) (読み取り専用)

my_btn で指定されているボタンインスタンスのターゲットパスを返します。

例

my_btn という名前のボタンインスタンスをステージに追加し、タイムラインのフレーム 1 に次のコードを追加します。

```
trace(my_btn._target); //displays /my_btn
```

my_btn を選択し、ムービークリップに変換します。新しいムービークリップに my_mc というインスタンス名をつけます。タイムラインのフレーム 1 の既存の ActionScript を削除し、次のコードで置き換えます。

```
my_mc.my_btn.onRelease = function(){  
    trace(this._target); //displays /my_mc/my_btn  
};
```

スラッシュ表記をドット表記に変換するには、上記のコードを次のように変更します。

```
my_mc.my_btn.onRelease = function(){  
    trace(eval(this._target)); //displays _level0.my_mc.my_btn  
};
```

これにより、次のようにターゲットオブジェクトのメソッドやパラメータにアクセスできるようになります。

```
my_mc.my_btn.onRelease = function(){
    var target_btn:Button = eval(this._target);
    trace(target_btn._name); //displays my_btn
};
```

関連項目

[_target \(MovieClip._target プロパティ\)](#)

trackAsMenu (Button.trackAsMenu プロパティ)

public trackAsMenu : Boolean

他のボタンまたはムービークリップがマウスまたはスタイルスから解放イベントを受け取ることができるかどうかを示す布尔値です。ボタンを横切ってスタイルスまたはマウスポインタをドラッグし、2番目のボタン上でボタンを離すと、onRelease イベントが2番目のボタンに対して登録されます。これを使用して、2番目のボタンのメニューを作成できます。trackAsMenu プロパティは、任意のボタンまたはムービークリップオブジェクトで設定できます。trackAsMenu プロパティを定義していない場合、デフォルトの動作は false です。

trackAsMenu プロパティは必要に応じていつでも変更できます。変更は即座に反映されます。

メモ: Flash Lite 2.0 では、trackAsMenu プロパティは、System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、2つのボタンをメニューとして追跡する方法を示します。ステージ上に one_btn と two_btn の2つのボタンインスタンスを配置します。タイムラインに次の ActionScript を配置します。

```
var one_btn:Button;
var two_btn:Button;
one_btn.trackAsMenu = true;
two_btn.trackAsMenu = true
one_btn.onRelease = function() {
    trace("clicked one_btn");
};
two_btn.onRelease = function() {
    trace("clicked two_btn");
};
```

SWF ファイルをテストするには、ステージの one_btn 上でクリックし、マウスボタンを押したまま two_btn 上に移動した後でマウスボタンを離します。次に、ActionScript の中で trackAsMenu を含む 2 行をコメントアウトします。再び SWF ファイルをテストして、ボタンの動作の違いを確認します。

関連項目

[trackAsMenu \(MovieClip.trackAsMenu プロパティ \)](#)

_url (Button._url プロパティ)

public _url : String (読み取り専用)

ボタンの配置された SWF ファイルの URL を取得します。

例

ステージ上に one_btn と two_btn の 2 つのボタンインスタンスを作成します。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var one_btn:Button;
var two_btn:Button;
this.createTextField("output_txt", 999, 0, 0, 100, 22);
output_txt.autoSize = true;
one_btn.onRelease = function() {
    trace("clicked one_btn");
    trace(this._url);
};
two_btn.onRelease = function() {
    trace("clicked "+this._name);
    var url_array:Array = this._url.split("/");
    var my_str:String = String(url_array.pop());
    output_txt.text = unescape(my_str);
};
```

それぞれのボタンをクリックすると、ボタンが含まれる SWF のファイル名が [出力] パネルまたは TextField インスタンスに表示されます。

_visible (Button._visible プロパティ)

public _visible : Boolean

my_btn で指定されているボタンを表示するかどうかを示す布尔値です。_visible プロパティが false に設定されている非表示のボタンは、使用できません。

例

ステージ上に 2 つのボタンを作成し、それぞれのインスタンスに myBtn1_btn と myBtn2_btn という名前を付けます。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
myBtn1_btn.onRelease = function() {
    this._visible = false;
    trace("clicked "+this._name);
};
myBtn2_btn.onRelease = function() {
```

```
this._alpha = 0;
trace("clicked "+this._name);
};

アルファを 0 に設定した後でも myBtn2_btn をクリックできます。
```

関連項目

[_visible \(MovieClip._visible プロパティ\)](#), [_visible \(TextField._visible プロパティ\)](#)

_width (Button._width プロパティ)

public _width : Number

ボタンの幅(ピクセル単位)です。

例

次の例では、ボタン my_btn の width プロパティの値を大きくして、幅を [出力] パネルに表示します。タイムラインのフレーム1に次の ActionScript を入力します。

```
my_btn.onRelease = function() {
    trace(this._width);
    this._width ~= 1.1;
};
```

関連項目

[_width \(MovieClip._width プロパティ\)](#)

_x (Button._x プロパティ)

public _x : Number

親ムービークリップのローカル座標を基準にしたボタンの x 座標を設定する整数です。メインのタイムラインにあるムービークリップの座標系は、ステージの左上隅を (0,0) として参照します。変形されているムービークリップ内にあるボタンの座標系は、ボタンを囲むムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップ内のボタンは、反時計回りに 90 度回転した座標系を継承します。ボタンの座標は、基準点の位置を参照します。

例

次の例では、ステージ上の my_btn の座標を 0 に設定します。ボタン my_btn を作成し、タイムラインのフレーム1に次の ActionScript を入力します。

```
my_btn._x = 0;
my_btn._y = 0;
```

関連項目

[_xscale \(Button._xscale プロパティ\)](#), [_y \(Button._y プロパティ\)](#), [_yscale \(Button._yscale プロパティ\)](#)

_xmouse (Button._xmouse プロパティ)

public _xmouse : Number (読み取り専用)

hasMouse が true の場合にボタンを基準にしたマウス位置の x 座標を返します。

メモ : Flash Lite 2.0 では、_xmouse プロパティは、System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、ステージのそれぞれ基準とした x 座標と、ステージ上に配置されたボタン my_btn を表示します。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'>";
    table_str += "<b>Stage</b>\t"+_xmouse+"\t"+_ymouse+newline;
    table_str += "<b>Button</b>\t"+_xmouse+"\t"+_ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

関連項目

[_ymouse \(Button._ymouse プロパティ\)](#)

_xscale (Button._xscale プロパティ)

public _xscale : Number

ボタンの基準点から適用した場合のボタンの水平方向の拡大 / 縮小率(パーセント単位)です。デフォルトの基準点は(0,0)です。

ローカル座標系を拡大または縮小すると、_x プロパティと _y プロパティの設定に影響します。この設定はピクセル単位で表されます。たとえば、親ムービークリップを 50% に縮小して、_x プロパティを設定すると、ボタン内のオブジェクトの移動距離は、拡大 / 縮小率が 100% だったときの半分のピクセル数になります。

例

次の例では、ボタン my_btn を拡大します。ボタンをクリックして離すと、ボタンは x 軸および y 軸方向に 10% ずつ拡大されます。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
my_btn.onRelease = function(){
    this._xscale ~= 1.1;
    this._yscale ~= 1.1;
};
```

関連項目

[_x \(Button._x プロパティ \), \[_y \\(Button._y プロパティ \\), \\[_yscale \\\(Button._yscale プロパティ \\\)\\]\\(#\\)\]\(#\)](#)

_y (Button._y プロパティ)

public _y : Number

親ムービークリップのローカル座標を基準にしたボタンの y 座標です。メインのタイムラインにあるボタンの座標系は、ステージの左上隅を(0,0)として参照します。変形されている別のムービークリップ内にあるボタンの座標系は、ボタンを囲むムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップ内のボタンは、反時計回りに 90 度回転した座標系を継承します。ボタンの座標は、基準点の位置を参照します。

例

次の例では、ステージ上の my_btn の座標を 0 に設定します。ボタン my_btn を作成し、タイムラインのフレーム 1 に次の ActionScript を入力します。

```
my_btn._x = 0;
my_btn._y = 0;
```

関連項目

[_x \(Button._x プロパティ \), \[_xscale \\(Button._xscale プロパティ \\), \\[_yscale \\\(Button._yscale プロパティ \\\)\\]\\(#\\)\]\(#\)](#)

_ymouse (Button._ymouse プロパティ)

public _ymouse : Number (読み取り専用)

ボタンを基準にしたマウス位置のy座標を返します。

メモ : Flash Lite 2.0 では、_ymouse プロパティは、System.capabilities.hasMouse が true または System.capabilities.hasStylus が true の場合にのみサポートされます。

例

次の例では、ステージのそれぞれ基準としたy座標と、ステージ上に配置されたボタン my_btn を表示します。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'>";
    table_str += "<b>Stage</b>\t"+x:_xmouse+"\t"+y:_ymouse+newline;
    table_str += "<b>Button</b>\t"+x:my_btn._xmouse+"\t"+y:my_btn._ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

関連項目

[_xmouse \(Button._xmouse プロパティ \)](#)

_yscale (Button._yscale プロパティ)

public _yscale : Number

ボタンの基準点から適用した場合のボタンの垂直方向の拡大 / 縮小率 (パーセント単位) です。デフォルトの基準点は (0,0) です。

例

次の例では、ボタン my_btn を拡大します。ボタンをクリックして離すと、ボタンは x 軸および y 軸方向に 10% ずつ拡大されます。タイムラインのフレーム 1 に次の ActionScript を入力します。

```
my_btn.onRelease = function(){
    this._xscale ~= 1.1;
    this._yscale ~ 1.1;
};
```

関連項目

[_y \(Button._y プロパティ\)](#), [_x \(Button._x プロパティ\)](#), [_xscale \(Button._xscale プロパティ\)](#)

capabilities (System.capabilities)

```
Object
|
+-System.capabilities
```

```
public class capabilities
extends Object
```

Capabilities クラスを使用すると、SWF ファイルのホストであるシステムと Flash Player の機能を確認できるので、コンテンツをさまざまな形式に合わせることができます。たとえば、モバイルデバイスの画面はコンピュータの画面とは異なります。できるだけ多くのユーザーに適切なコンテンツを提供するため、System.capabilities オブジェクトを使用して、各ユーザーが使用しているデバイスのタイプを確認できます。デバイスの機能に応じて異なる SWF ファイルを送るようにサーバーに指示したり、デバイスの機能に応じて表示形式を変更するように SWF ファイルに指示したりできます。

機能情報は、HTTP の GET メソッドまたは POST メソッドを使用して送信できます。

次の例は、モバイルデバイスが以下の状態であることを示すストリングです。

- 標準の画面の向きが指定されている
- 判別できない言語を実行している
- Symbian7.0sSeries60V2 オペレーティングシステムを実行している
- ユーザーがハードディスク、カメラ、マイクにアクセスできないように設定されている
- Flash Lite プレーヤーが公式リリースバージョンである
- Flash Lite プレーヤーが Flash Media Server を通じて実行されるスクリーンブロードキャスト アプリケーションの開発や再生をサポートしていない
- デバイスでの印刷をサポートしていない
- Flash Lite プレーヤーが埋め込みビデオをサポートするモバイルデバイスで実行されている

```
undefinedScreenOrientation=normal language=xu OS=Symbian7.0sSeries60V2
```

```
localFileReadDisable=true avHardwareDisable=true isDebugger=false
```

```
hasScreenBroadcast=false hasScreenPlayback=false hasPrinting=false
```

```
hasEmbeddedVideo=true
```

System.capabilities オブジェクトのプロパティのほとんどは読み取り専用です。

プロパティ一覧

オプション	プロパティ	説明
static	<code>audioMIMETypes:Array</code> (読み取り専用)	モバイルデバイスでサポートされているオーディオコーデックの MIME タイプの配列を返します。
static	<code>avHardwareDisable:Boolean</code> (読み取り専用)	ユーザーのカメラとマイクへのアクセスが管理上禁止されているか(<code>true</code>)許可されているか(<code>false</code>)を示すブール値です。
static	<code>has4WayKeyAS:Boolean</code> (読み取り専用)	Flash Lite プレーヤーが、左、右、上、下方向キーに関連付けられたキーイベントハンドラの ActionScript コードを実行する場合は <code>true</code> になるブール値です。
static	<code>hasAccessibility:Boolean</code> (読み取り専用)	Flash Player が実行されている環境が、Flash Player とアクセシビリティ補助との間の通信をサポートしている場合は <code>true</code> 、サポートしていない場合は <code>false</code> になるブール値です。
static	<code>hasAudio:Boolean</code> (読み取り専用)	システムにオーディオ機能があるかどうかを示します。
static	<code>hasAudioEncoder:Boolean</code> (読み取り専用)	Flash Player がオーディオストリームをエンコードできるかどうかを示します。
static	<code>hasCMIDI:Boolean</code> (読み取り専用)	モバイルデバイスが CMIDI オーディオ形式のサウンドデータを再生できる場合は <code>true</code> を返します。
static	<code>hasCompoundSound:Boolean</code> (読み取り専用)	Flash Lite プレーヤーがコンパウンドサウンドデータを処理できる場合は <code>true</code> を返します。
static	<code>hasDataLoading:Boolean</code> (読み取り専用)	Flash Lite プレーヤーが特定の関数の呼び出しによって追加データを動的にロードできる場合は <code>true</code> を返します。
static	<code>hasEmail:Boolean</code> (読み取り専用)	Flash Lite プレーヤーが GetURL ActionScript コマンドを使用して電子メールメッセージを送信できる場合は <code>true</code> を返します。
static	<code>hasEmbeddedVideo:Boolean</code> (読み取り専用)	モバイルデバイスが埋め込みビデオをサポートしているかどうかを示すブール値です。
static	<code>hasMappableSoftKeys:Boolean</code> (読み取り専用)	モバイルデバイスでソフトキーのラベルをリセットまたは再割り当てし、変更したソフトキーからイベントを処理できる場合は <code>true</code> を返します。
static	<code>hasMFI:Boolean</code> (読み取り専用)	モバイルデバイスが MFI オーディオ形式のサウンドデータを再生できる場合は <code>true</code> を返します。

オプション	プロパティ	説明
static	<code>hasMIDI:Boolean</code> (読み取り専用)	モバイルデバイスが MIDI オーディオ形式のサウンドデータを再生できる場合は <code>true</code> を返します。
static	<code>hasMMS:Boolean</code> (読み取り専用)	モバイルデバイスが GetURL ActionScript コマンドを使用して MMS メッセージを送信できる場合は <code>true</code> を返します。
static	<code>hasMouse:Boolean</code> (読み取り専用)	モバイルデバイスがマウス関連のイベントを Flash Lite プレーヤーに送信できるかどうかを示します。
static	<code>hasMP3:Boolean</code> (読み取り専用)	モバイルデバイスに MP3 デコーダがあるかどうかを示します。
static	<code>hasPrinting:Boolean</code> (読み取り専用)	Flash Player が実行されているモバイルデバイスが印刷機能をサポートしている場合は <code>true</code> 、サポートしない場合は <code>false</code> になるブール値です。
static	<code>hasQWERTYKeyboard:Boolean</code> (読み取り専用)	Backspace キーなど、標準的な QWERTY キーボードにあるすべてのキーに関連付けられた ActionScript コードを Flash Lite プレーヤーが処理できる場合は <code>true</code> を返します。
static	<code>hasScreenBroadcast:Boolean</code> (読み取り専用)	Flash Media Server を通じて実行されるスクリーンブロードキャストアプリケーションの開発を Flash Player がサポートしている場合は <code>true</code> 、サポートしていない場合は <code>false</code> になるブール値です。
static	<code>hasScreenPlayback:Boolean</code> (読み取り専用)	Flash Media Server を通じて実行されるスクリーンブロードキャストアプリケーションの再生を Flash Player がサポートしている場合は <code>true</code> 、サポートしていない場合は <code>false</code> になるブール値です。
static	<code>hasSharedObjects:Boolean</code> (読み取り専用)	アプリケーションで再生される Flash Lite コンテンツが Flash Lite バージョンの共有オブジェクトにアクセスできる場合は <code>true</code> を返します。
static	<code>hasSMAF:Boolean</code> (読み取り専用)	モバイルデバイスが SMAF オーディオ形式のサウンドデータを再生できる場合は <code>true</code> を返します。
static	<code>hasSMS:Number</code> (読み取り専用)	モバイルデバイスが GetURL ActionScript コマンドを使用して SMS メッセージを送信できるかどうかを示します。
static	<code>hasStreamingAudio:Boolean</code> (読み取り専用)	プレーヤーがストリーミングオーディオを再生できる場合は <code>true</code> 、再生できない場合は <code>false</code> になるブール値です。
static	<code>hasStreamingVideo:Boolean</code> (読み取り専用)	プレーヤーがストリーミングビデオを再生できるかどうかを示すブール値です。

オプション	プロパティ	説明
static	<code>hasStylus:Boolean</code> (読み取り専用)	モバイルデバイスがスタイルス関連のイベントをサポートしているかどうかを示します。
static	<code>hasVideoEncoder:Boolean</code> (読み取り専用)	Flash Player がビデオストリームをエンコードできるかどうかを示します。
static	<code>hasXMLSocket:Number</code> (読み取り専用)	ホストアプリケーションが XML ソケットをサポートしているかどうかを示します。
static	<code>imageMIMETypes:Array</code> (読み取り専用)	<code>loadMovie</code> 関数およびモバイルデバイスのコードックがイメージを処理するためにサポートしているすべての MIME タイプを含む配列を返します。
static	<code>isDebugger:Boolean</code> (読み取り専用)	Flash Player が正式にリリースされたバージョンか (<code>false</code>) デバッグ用の特別なバージョンか (<code>true</code>) を示すブール値です。
static	<code>language:String</code> (読み取り専用)	Flash Player が実行されているシステムの言語を示します。
static	<code>localFileReadDisable:Boolean</code> (読み取り専用)	ユーザーのハードディスクへの読み取りアクセスが管理上禁止されているか (<code>true</code>)、許可されているか (<code>false</code>) を示すブール値です。
static	<code>MIMETypes:Array</code> (読み取り専用)	<code>loadMovie</code> 関数、 <code>Sound</code> オブジェクト、および <code>Video</code> オブジェクトがサポートするすべての MIME タイプを含む配列を返します。
static	<code>os:String</code> (読み取り専用)	現在のオペレーティングシステムを示すストリングです。
static	<code>screenOrientation:String</code> (読み取り専用)	<code>System.capabilities</code> オブジェクトのこのメンバー変数は現在の画面の向きを示します。
static	<code>screenResolutionX:Number</code> (読み取り専用)	画面の最大水平解像度を示す整数です。
static	<code>screenResolutionY:Number</code> (読み取り専用)	画面の最大垂直解像度を示す整数です。
static	<code>softKeyCount:Number</code> (読み取り専用)	モバイルデバイスがサポートしている再マップ可能なソフトキーの数を示します。

オプション	プロパティ	説明
static	<code>version:String</code> (読み取り専用)	Flash Player のプラットフォームとバージョンの情報 (たとえば "WIN 7,1,0,0" など) が含まれるストリングです。
static	<code>videoMIMETypes:Array</code> (読み取り専用)	モバイルデバイスのコーデックがサポートしているビデオのすべての MIME タイプを示します。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

メソッド一覧

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty (Object.hasOwnProperty メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf メソッド), registerClass (Object.registerClass メソッド), toString (Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf (Object.valueOf メソッド), watch (Object.watch メソッド)
```

audioMIMETypes (capabilities.audioMIMETypes プロパティ)

`public static audioMIMETypes : Array` (読み取り専用)

モバイルデバイスでサポートされているオーディオコーデックの MIME タイプの配列を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.audioMIMETypes);
```

avHardwareDisable (capabilities.avHardwareDisable プロパティ)

```
public static avHardwareDisable : Boolean (読み取り専用)
```

ユーザーのカメラとマイクへのアクセスが管理上禁止されているか(true)許可されているか(false)を示すブール値です。サーバーストリングはAVDです。

メモ : Flash Lite 2.0 では、戻り値は常に true です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.avHardwareDisable);
```

has4WayKeyAS (capabilities.has4WayKeyAS プロパティ)

```
public static has4WayKeyAS : Boolean (読み取り専用)
```

Flash Lite プレーヤーが、左、右、上、下方向キーに関連付けられたキーイベントハンドラの ActionScript コードを実行する場合は true になるブール値です。それ以外の場合は、このプロパティは false を返します。

この変数の値が true の場合、4 方向のキーのいずれかが押されると、プレーヤーは最初にそのキーのハンドラを探します。ハンドラが見つからない場合、Flash はコントロールナビゲーションを実行します。しかし、イベントハンドラが見つかった場合は、そのキーのナビゲーションアクションは発生しません。つまり、下矢印キーに対して keypress ハンドラがある場合、下方向に移動する機能が無効になります。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.has4WayKeyAS);
```

hasAccessibility (capabilities.hasAccessibility プロパティ)

```
public static hasAccessibility : Boolean (読み取り専用)
```

Flash Player が実行されている環境が、Flash Player とアクセシビリティ補助との間の通信をサポートしている場合は true、サポートしていない場合は false になるブール値です。サーバーストリングは ACC です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasAccessibility);
```

hasAudio (capabilities.hasAudio プロパティ)

public static hasAudio : Boolean (読み取り専用)

システムにオーディオ機能があるかどうかを示します。Flash Player が実行されているシステムがオーディオ機能をサポートしている場合は true、サポートしない場合は false になる布尔値です。サーバーストリングは A です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasAudio);
```

hasAudioEncoder (capabilities.hasAudioEncoder プロパティ)

public static hasAudioEncoder : Boolean (読み取り専用)

Flash Player がオーディオストリームをエンコードできるかどうかを示します。Flash Player がオーディオストリーム（マイクからのストリームなど）をエンコードできる場合は true、エンコードできない場合は false になる布尔値です。サーバーストリングは AE です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasAudioEncoder);
```

hasCMIDI (capabilities.hasCMIDI プロパティ)

public static hasCMIDI : Boolean (読み取り専用)

モバイルデバイスが CMIDI オーディオ形式のサウンドデータを再生できる場合は true を返します。再生できない場合、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasCMIDI);
```

hasCompoundSound

(capabilities.hasCompoundSound プロパティ)

public static hasCompoundSound : Boolean (読み取り専用)

Flash Lite プレーヤーがコンパウンドサウンドデータを処理できる場合は true を返します。それ以外の場合は、false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasCompoundSound);
```

hasDataLoading (capabilities.hasDataLoading プロパティ)

public static hasDataLoading : Boolean (読み取り専用)

Flash Lite プレーヤーが特定の関数の呼び出しによって追加データを動的にロードできる場合は true を返します。

次の関数を呼び出すことができます。

- loadMovie()
- loadMovieNum()
- loadVariables()
- loadVariablesNum()
- XML.parseXML()
- Sound.loadSound()
- MovieClip.loadVariables()
- MovieClip.loadMovie()
- MovieClipLoader.loadClip()
- LoadVars.load()
- LoadVars.sendAndLoad()

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasDataLoading);
```

hasEmail (capabilities.hasEmail プロパティ)

public static hasEmail : Boolean (読み取り専用)

Flash Lite プレーヤーが GetURL ActionScript コマンドを使用して電子メールメッセージを送信できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasEmail);
```

hasEmbeddedVideo

(capabilities.hasEmbeddedVideo プロパティ)

public static hasEmbeddedVideo : Boolean (読み取り専用)

モバイルデバイスが埋め込みビデオをサポートしているかどうかを示す布尔値です。

メモ : hasEmbeddedVideo プロパティは、Flash Lite 2.0 および Flash Lite 2.1 では常に true となり、ライブラリがデバイスピデオをサポートしていることを示します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasEmbeddedVideo);
```

hasMappableSoftKeys

(capabilities.hasMappableSoftKeys プロパティ)

public static hasMappableSoftKeys : Boolean

モバイルデバイスでソフトキーのラベルをリセットまたは再割り当てし、変更したソフトキーからイベントを処理できる場合は true を返します。それ以外の場合は、false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMappableSoftKeys);
```

hasMFI (capabilities.hasMFI プロパティ)

public static hasMFI : Boolean (読み取り専用)

モバイルデバイスが MFI オーディオ形式のサウンドデータを再生できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMFI);
```

hasMIDI (capabilities.hasMIDI プロパティ)

public static hasMIDI : Boolean (読み取り専用)

モバイルデバイスが MIDI オーディオ形式のサウンドデータを再生できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMIDI);
```

hasMMS (capabilities.hasMMS プロパティ)

public static hasMMS : Boolean (読み取り専用)

モバイルデバイスが GetURL ActionScript コマンドを使用して MMS メッセージを送信できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMMS);
```

hasMouse (capabilities.hasMouse プロパティ)

public static hasMouse : Boolean (読み取り専用)

モバイルデバイスがマウス関連のイベントを Flash Lite プレーヤーに送信できるかどうかを示します。

モバイルデバイスがマウス関連のイベントを Flash Lite プレーヤーに送信できる場合、このプロパティは true を返します。それ以外の場合は、false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMouse);
```

hasMP3 (capabilities.hasMP3 プロパティ)

public static hasMP3 : Boolean (読み取り専用)

モバイルデバイスに MP3 デコーダがあるかどうかを示します。Flash Player が実行されているシステムに MP3 デコーダがある場合は true、MP3 デコーダがない場合は false になる布尔値です。サーバーストリングは MP3 です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasMP3);
```

hasPrinting (capabilities.hasPrinting プロパティ)

public static hasPrinting : Boolean (読み取り専用)

Flash Player が実行されているモバイルデバイスが印刷機能をサポートしている場合は true、サポートしない場合は false になる布尔値です。サーバーストリングは PR です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasPrinting);
```

hasQWERTYKeyboard (capabilities.hasQWERTYKeyboard プロパティ)

public static hasQWERTYKeyboard : Boolean (読み取り専用)

Backspace キーなど、標準的な QWERTY キーボードにあるすべてのキーに関連付けられた ActionScript コードを Flash Lite プレーヤーが処理できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasQWERTYKeyboard);
```

hasScreenBroadcast (capabilities.hasScreenBroadcast プロパティ)

public static hasScreenBroadcast : Boolean (読み取り専用)

Flash Media Server を通じて実行されるスクリーンブロードキャストアプリケーションの開発を Flash Player がサポートしている場合は true、サポートしていない場合は false になる布尔値です。サーバーストリングは SB です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasScreenBroadcast);
```

hasScreenPlayback (capabilities.hasScreenPlayback プロパティ)

public static hasScreenPlayback : Boolean (読み取り専用)

Flash Media Server を通じて実行されるスクリーンブロードキャストアプリケーションの再生を Flash Player がサポートしている場合は true、サポートしていない場合は false になる布尔値です。サーバーストリングは SP です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasScreenPlayback);
```

hasSharedObjects (capabilities.hasSharedObjects プロパティ)

public static hasSharedObjects : Boolean (読み取り専用)

アプリケーションで再生される Flash Lite コンテンツが Flash Lite バージョンの共有オブジェクトにアクセスできる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasSharedObjects);
```

hasSMAF (capabilities.hasSMAF プロパティ)

public static hasSMAF : Boolean (読み取り専用)

モバイルデバイスが SMAF オーディオ形式のサウンドデータを再生できる場合は true を返します。

それ以外の場合は、このプロパティは false を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasSMAF);
```

hasSMS (capabilities.hasSMS プロパティ)

public static hasSMS : Number (読み取り専用)

モバイルデバイスが GetURL ActionScript コマンドを使用して SMS メッセージを送信できるかどうかを示します。

Flash Lite が SMS メッセージを送信できる場合は、この変数が定義され、値が 1 に設定されます。それ以外の場合、この変数は定義されません。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasSMS);
```

hasStreamingAudio (capabilities.hasStreamingAudio プロパティ)

public static hasStreamingAudio : Boolean (読み取り専用)

プレーヤーがストリーミングオーディオを再生できる場合は true、再生できない場合は false になるブール値です。サーバーストリーミングは SA です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasStreamingAudio);
```

hasStreamingVideo (capabilities.hasStreamingVideo プロパティ)

public static hasStreamingVideo : Boolean (読み取り専用)

プレーヤーがストリーミングビデオを再生できるかどうかを示すブール値です。

メモ: hasStreamingVideo プロパティは、Flash Lite 2.0 および Flash Lite 2.1 では常に false となり、ストリーミング FLV がサポートされていないことを示します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasStreamingVideo);
```

hasStylus (capabilities.hasStylus プロパティ)

public static hasStylus : Boolean (読み取り専用)

モバイルデバイスがスタイラス関連のイベントをサポートしているかどうかを示します。

モバイルデバイスのプラットフォームがスタイラス関連のイベントをサポートしていない場合、このプロパティは true を返します。それ以外の場合は、このプロパティは false を返します。

スタイラスは onMouseMove イベントをサポートしていません。この capabilities フラグを使用して、Flash コンテンツでモバイルデバイスのプラットフォームがこのイベントをサポートしているかどうかを確認できます。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasStylus);
```

hasVideoEncoder (capabilities.hasVideoEncoder プロパティ)

public static hasVideoEncoder : Boolean (読み取り専用)

Flash Player がビデオストリームをエンコードできるかどうかを示します。Flash Player がビデオストリーム (Web カメラからのストリームなど) をエンコードできる場合は true、エンコードできない場合は false になるブール値です。サーバーストリングは VE です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.hasVideoEncoder);
```

hasXMLSocket (capabilities.hasXMLSocket プロパティ)

public static hasXMLSocket : Number (読み取り専用)

ホストアプリケーションが XML ソケットをサポートしているかどうかを示します。

ホストアプリケーションが XML ソケットをサポートしている場合、この変数が値 1 で定義されます。そうでない場合は、この変数は定義されません。

imageMIMETypes (capabilities.imageMIMETypes プロパティ)

public static imageMIMETypes : Array (読み取り専用)

loadMovie 関数およびモバイルデバイスのコーデックがイメージを処理するためにサポートしているすべての MIME タイプを含む配列を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.imageMIMETypes);
```

isDebugger (capabilities.isDebugger プロパティ)

public static isDebugger : Boolean (読み取り専用)

Flash Player が正式にリリースされたバージョンか (false) デバッグ用の特別なバージョンか (true) を示すブール値です。サーバーストリングは DEB です。

メモ : Flash Lite 2.0 では、戻り値は常に false です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.isDebugEnabled);
```

language (capabilities.language プロパティ)

public static language : String (読み取り専用)

Flash Player が実行されているシステムの言語を示します。このプロパティは、ISO 639-1による小文字 2 文字の言語コードで指定されます。中国語については、簡体字と繁体字を識別するために ISO 3166 による大文字 2 文字の国コードサブタグが追加されます。言語自体の名前には英語タグが使用されます。たとえば、fr はフランス語を示します。

このプロパティは、Flash Player 7 で 2 つおりの変更が加えられました。まず、英語版のシステムで言語コードに国コードが含まれなくなりました。Flash Player 6 の場合、このプロパティはすべての英語のシステムで言語コードと 2 文字の国コード (サブタグ) の組み合わせ (en-US) を返します。英語のシステムで Flash Player 7 を使用した場合、このプロパティは言語コード (en) だけを返します。次に、Microsoft Windows システムの場合、このプロパティはユーザーインターフェイス (UI) 言語を返します。Microsoft Windows プラットフォーム上で Flash Player 6 を使用した場合、

System.capabilities.language は、ユーザーロケールを返します。ユーザーロケールは、日付、時間、通貨などをフォーマットするための設定を制御します。Microsoft Windows プラットフォーム上で Flash Player 7 を使用した場合、このプロパティは UI 言語を返します。UI 言語は、すべてのメニュー、ダイアログボックス、エラーメッセージ、およびヘルプファイルなどで使用される言語を参照します。

言語	タグ
チェコ語	cs
デンマーク語	da
オランダ語	nl
英語	en
フィンランド語	fi
フランス語	fr
ドイツ語	de

言語	タグ
ハンガリー語	hu
イタリア語	it
日本語	ja
韓国語	ko
ノルウェー語	no
その他 / 不明	xu
ポーランド語	pl
ポルトガル語	pt
ロシア語	ru
簡体字中国語	zh-CN
スペイン語	es
スウェーデン語	sv
繁体字中国語	zh-TW
トルコ語	tr

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.language);
```

localFileReadDisable

(capabilities.localFileReadDisable プロパティ)

public static localFileReadDisable : Boolean (読み取り専用)

ユーザーのハードディスクへの読み取りアクセスが管理上禁止されているか (true)、許可されているか (false) を示すブール値です。このプロパティが true に設定されている場合、Flash Player では、ユーザーのハードディスクからファイル (Flash Player が起動するときの最初の SWF ファイルを含む) を読み取ることができません。たとえば、XML.load()、LoadMovie()、または LoadVars.load() を使用してユーザーのハードディスク上にあるファイルを読み取ろうとしても、このプロパティが true の場合は、読み取りに失敗します。

このプロパティが true に設定されていると、ランタイム共有ライブラリも読み取ることができません。ただし、ローカル共有オブジェクトは、このプロパティの値に関係なく読み取ることができます。サーバースtringing は LFD です。

メモ : Flash Lite 2.0 では、戻り値は常に true です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.localFileReadDisable);
```

MIMETypes (capabilities.MIMETypes プロパティ)

public static MIMETypes : [Array](#) (読み取り専用)

loadMovie 関数、Sound オブジェクト、および Video オブジェクトがサポートするすべての MIME タイプを含む配列を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.MIMETypes);
```

os (capabilities.os プロパティ)

public static os : [String](#) (読み取り専用)

現在のオペレーティングシステムを示すストリングです。"os" プロパティの値は、"Windows XP"、"Windows 2000"、"Windows NT"、"Windows 98/ME"、"Windows 95"、"Windows CE"(デスクトップバージョンの Flash Player ではなく SDK のみ)、"Linux"、"MacOS" のいずれかです。サーバーストリングは OS です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.os);
```

screenOrientation (capabilities.screenOrientation プロパティ)

public static screenOrientation : [String](#) (読み取り専用)

System.capabilities オブジェクトのこのメンバー変数は現在の画面の向きを示します。

screenOrientation プロパティの値は次のいずれかです。

- normal。画面は通常の向きです。
- rotated90。画面は 90 度回転しています。
- rotated180。画面は 180 度回転しています。
- rotated270。画面は 270 度回転しています。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.screenOrientation);
```

screenResolutionX (capabilities.screenResolutionX プロパティ)

public static screenResolutionX : [Number](#) (読み取り専用)

画面の最大水平解像度を示す整数です。サーバーストリングは R で、画面の幅と高さの両方を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.screenResolutionX);
```

screenResolutionY (capabilities.screenResolutionY プロパティ)

public static screenResolutionY : [Number](#) (読み取り専用)

画面の最大垂直解像度を示す整数です。サーバーストリングは R で、画面の幅と高さの両方を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.screenResolutionY);
```

softKeyCount (capabilities.softKeyCount プロパティ)

public static softKeyCount : [Number](#) (読み取り専用)

モバイルデバイスがサポートしている再マップ可能なソフトキーの数を示します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.softKeyCount);
```

version (capabilities.version プロパティ)

public static version : [String](#) (読み取り専用)

Flash Player のプラットフォームとバージョンの情報(たとえば "WIN 7,1,0,0" など)が含まれるストリングです。サーバースtringing は V です。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.version);
```

videoMIMETypes (capabilities.videoMIMETypes プロパティ)

public static videoMIMETypes : [Array](#) (読み取り専用)

モバイルデバイスのコーデックがサポートしているビデオのすべての MIME タイプを示します。

このプロパティは、モバイルデバイスのコーデックがサポートしているビデオのすべての MIME タイプの配列を返します。

例

次の例では、この読み取り専用プロパティの値をトレースします。

```
trace(System.capabilities.videoMIMETypes);
```

Color

[Object](#)

```
|  
+-Color
```

```
public class Color  
extends Object
```

Color クラスを使用すると、ムービークリップの RGB カラー値とカラー変換を設定したり、設定した値を取得したりできます。

Color オブジェクトのメソッドを呼び出すには、コンストラクタ new Color() を使用して Color オブジェクトを作成する必要があります。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

コンストラクター一覧

シグネチャ	説明
Color (target:Object)	target_mc パラメータで指定されたムービークリップ用の Color オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	getRGB() : Number	Color オブジェクトで使用中の R+G+B 組み合わせを返します。
	getTransform() : Object	Color.setTransform() の前回の呼び出しで設定されたカラー変換情報を返します。
	setRGB(offset: Number) : Void	Color オブジェクトの RGB カラーを指定します。
	setTransform(transformObject: Object) : Void	Color オブジェクトのカラー変換情報を設定します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

Color コンストラクタ

```
public Color(target:Object)
```

target_mc パラメータで指定されたムービークリップ用の Color オブジェクトを作成します。作成した Color オブジェクトのメソッドを使用して、ターゲットムービークリップ全体の色を変更できます。

パラメータ

target: Object - ムービークリップのインスタンス名。

例

次の例では、ムービークリップ *my_mc* 用に *my_color* という Color オブジェクトを作成し、その RGB 値をオレンジ色に設定します。

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xff9933);
```

getRGB (Color.getRGB メソッド)

```
public getRGB():Number
```

Color オブジェクトで使用中の R+G+B 組み合わせを返します。

戻り値

Number - 指定された色の RGB 値を表す数値。

例

次のコードでは、Color オブジェクト *my_color* の RGB 値を取得し、それを 16 進ストリングに変換して、変数 *myValue* に代入します。このコードの動作を確認するには、ムービークリップのインスタンスをステージに追加し、そのインスタンスに *my_mc* という名前を付けます。

```
var my_color:Color = new Color(my_mc);
// set the color
my_color.setRGB(0xff9933);
var myValue:String = my_color.getRGB().toString(16);
// trace the color value
trace(myValue); // traces ff9933
```

関連項目

[setRGB \(Color.setRGB メソッド \)](#)

getTransform (Color.getTransform メソッド)

public getTransform() : Object

Color.setTransform() の前回の呼び出しで設定されたカラー変換情報を返します。

戻り値

Object - 指定された色に関する現在のオフセット値とパーセント値をプロパティとして持つオブジェクト。

例

次の例では、変換オブジェクトを取得し、現在の値を基準に my_mc のカラーおよびアルファの新しいパーセントを設定します。このコードの動作を確認するには、my_mc というインスタンス名のマルチカラーのムービークリップをステージに配置します。さらに、次のコードをメインタイムラインのフレーム1に追加し、[制御]-[ムービープレビュー]を選択します。

```
var my_color:Color = new Color(my_mc);
var myTransform:Object = my_color.getTransform();
myTransform = { ra: 50, ba: 50, aa: 30};
my_color.setTransform(myTransform);
```

カラー変換オブジェクトのパラメータの詳細については、Color.setTransform() を参照してください。

関連項目

[setTransform \(Color.setTransform メソッド\)](#)

setRGB (Color.setRGB メソッド)

public setRGB(offset:Number) : Void

Color オブジェクトの RGB カラーを指定します。このメソッドを呼び出すと、Color.setTransform() のそれまでの設定が上書きされます。

パラメータ

offset:Number - 0xRRGGBB 設定される 16 進カラーまたは RGB カラー。RR、GG、BB はそれぞれ、各カラー成分のオフセットを指定する 2 衔の 16 進数で構成されます。0x により、ActionScript コンパイラは数値を 16 進数として解釈します。

例

次の例では、ムービークリップ my_mc に RGB カラー値を設定します。このコードの動作を確認するには、my_mc というインスタンス名のムービークリップをステージに配置します。さらに、次のコードをメインタイムラインのフレーム 1 に追加し、[制御]-[ムービープレビュー] を選択します。

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xFF0000); // my_mc turns red
```

関連項目

[setTransform \(Color.setTransform メソッド\)](#)

setTransform (Color.setTransform メソッド)

public setTransform(transformObject:Object) : Void

Color オブジェクトのカラー変換情報を設定します。colorTransformObject パラメータは、new Object コンストラクタで作成する汎用オブジェクトです。このオブジェクトには、カラーの赤、緑、青、およびアルファ（透明度）の各成分のパーセント値およびオフセット値を指定するパラメータがあります。値の入力形式は 0xRRGGBBAA です。

カラー変換オブジェクトのパラメータは、[拡張効果] ダイアログボックスの設定に対応しており、次のように定義します。

- *ra* は、赤の成分のパーセント (-100 ~ 100) です。
- *rb* は、赤の成分のオフセット (-255 ~ 255) です。
- *ga* は、緑の成分のパーセント (-100 ~ 100) です。
- *gb* は、緑の成分のオフセット (-255 ~ 255) です。
- *ba* は、青の成分のパーセント (-100 ~ 100) です。
- *bb* は、青の成分のオフセット (-255 ~ 255) です。
- *aa* は、アルファのパーセント (-100 ~ 100) です。
- *ab* は、アルファのオフセット (-255 ~ 255) です。

colorTransformObject パラメータは次のように作成します。

```
var myColorTransform:Object = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

次のシンタックスを使用して、colorTransformObject パラメータを作成することもできます。

```
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb: 90,
                               aa: 40, ab: 70}
```

パラメータ

`transformObject:Object` - new Object コンストラクタで作成したオブジェクト。この Object クラスのインスタンスには、カラー変換情報を指定するプロパティ (ra、rb、ga、gb、ba、bb、aa、ab) が必要です。各プロパティについて以下に例示します。

例

次の例では、ターゲット SWF ファイル用に新しい Color オブジェクトを作成し、上で定義したプロパティを使用して汎用オブジェクト `myColorTransform` を作成します。次に、`setTransform()` メソッドを使用して `colorTransformObject` を Color オブジェクトに渡します。このコードを Flash (FLA) ドキュメントで使用するには、コードをメインタイムラインのフレーム 1 に追加し、`my_mc` というインスタンス名のムービークリップをステージに配置します。

```
// Create a color object called my_color for the target my_mc
var my_color:Color = new Color(my_mc);
// Create a color transform object called myColorTransform using
// Set the values for myColorTransform
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb: 90,
    aa: 40, ab: 70};
// Associate the color transform object with the Color object
// created for my_mc
my_color.setTransform(myColorTransform);
```

関連項目

[Object](#)

Date

```
Object
|
+-Date
```

```
public class Date
extends Object
```

`Date` クラスを使用すると、世界時 (グリニッジ標準時、現在の呼称は世界標準時または UTC) を基準にするか、Flash Player を実行しているオペレーティングシステムを基準にして、日時の値調べることができます。`Date` クラスのメソッドは静的ではありません。メソッドを呼び出すときに指定した個々の `Date` オブジェクトにのみ適用されます。`Date.UTC()` メソッドは例外で、これは静的なメソッドです。

Date クラスによる夏時間の処理方法は、オペレーティングシステムおよび Flash Player のバージョンに応じて異なります。Flash Player 6 以降では、夏時間は各オペレーティングシステムで次のように処理されます。

- Windows - Date オブジェクトの出力は夏時間に合わせて自動的に調整されます。Date オブジェクトは、夏時間が現在の地域で採用されているかどうかを確認します。夏時間が採用されている場合は、夏時間に移行する日付と時刻の基準を確認します。ただし、現在有効な夏時間の日付が過去と未来に適用されるために、夏時間の日付が異なる地域では、計算される夏時間の日付が過去の日付と食い違う場合があります。
- Mac OS X - Date オブジェクトの出力は夏時間に合わせて自動的に調整されます。Mac OS X では、タイムゾーン情報データベースを使用して、現在または過去の日付または時刻に夏時間の時差を適用する必要があるかどうかを決定します。
- Mac OS 9 - 現在の日付および時刻に夏時間の時差を適用する必要があるかどうかを判定できるだけの情報しか提供されません。したがって、Date オブジェクトは、現在の夏時間の時差が過去および将来のすべての日時に適用されると想定します。

Flash Player 5 によるオペレーティングシステム別の夏時間の対処方法は次のとおりです。

- Windows - 夏時間に関する米国の規則が常に適用されるために、米国とは夏時間に移行する時期が異なるヨーロッパおよび他の地域では移行期間が食い違います。Flash は、現在の地域で採用されている夏時間を正しく検出します。

Date クラスのメソッドを呼び出すには、まず後述の Date クラスのコンストラクタを使用して Date オブジェクトを作成する必要があります。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>Date ([yearOrTimevalue: Number], [month:Number], [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number])</code>	指定された日時を保持する新しい Date オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>getDate() : Number</code>	指定された Date オブジェクトの日付(1～31の整数)をローカル時間で返します。
	<code>getDay() : Number</code>	指定された Date オブジェクトの曜日(日曜日は0、月曜日は1など)をローカル時間で返します。
	<code>getFullYear() : Number</code>	指定された Date オブジェクトの年(2000など、4桁の数字)をローカル時間で返します。
	<code>getHours() : Number</code>	指定された Date オブジェクトの時(0～23の整数)をローカル時間で返します。
	<code>getLocaleLongDate() : String</code>	現在定義されているロケールに従ってフォーマットされた、長い形式の現在日付を表すストリングを返します。
	<code>getLocaleShortDate() : String</code>	現在定義されているロケールに従ってフォーマットされた、短い形式の現在日付を表すストリングを返します。
	<code>getLocaleTime() : String</code>	現在定義されているロケールに従ってフォーマットされた現在時刻を表すストリングを返します。
	<code>getMilliseconds() : Number</code>	指定された Date オブジェクトのミリ秒(0～999の整数)をローカル時間で返します。
	<code>getMinutes() : Number</code>	指定された Date オブジェクトの分(0～59の整数)をローカル時間で返します。
	<code>getMonth() : Number</code>	指定された Date オブジェクトの月(1月は0、2月は1など)をローカル時間で返します。
	<code>getSeconds() : Number</code>	指定された Date オブジェクトの秒(0～59の整数)をローカル時間で返します。

オプション	シグネチャ	説明
	<code>getTime() : Number</code>	指定された Date オブジェクトに関して、1970 年 1 月 1 日 0 時(世界時)からのミリ秒数を返します。
	<code>getTimezoneOffset() : Number</code>	コンピュータのローカル時間と世界時の差(分単位)を返します。
	<code>getUTCDate() : Number</code>	指定された Date オブジェクトの日付(1~31 の整数)を世界時で返します。
	<code>getUTCDay() : Number</code>	指定された Date オブジェクトの曜日(日曜日は 0、月曜日は 1 など)を世界時で返します。
	<code>getUTCFullYear() : Number</code>	指定された Date オブジェクトの年(4 衡表記)を世界時で返します。
	<code>getUTCHours() : Number</code>	指定された Date オブジェクトの時(0~23 の整数)を世界時で返します。
	<code>getUTCMilliseconds() : Number</code>	指定された Date オブジェクトのミリ秒(0~999 の整数)を世界時で返します。
	<code>getUTCMinutes() : Number</code>	指定された Date オブジェクトの分(0~59 の整数)を世界時で返します。
	<code>getUTCMonth() : Number</code>	指定された Date オブジェクトの月(0[1月]~11[12月])を世界時で返します。
	<code>getUTCSeconds() : Number</code>	指定された Date オブジェクトの秒(0~59 の整数)を世界時で返します。
	<code>getUTCYear() : Number</code>	この Date の年を世界時(UTC)で返します。
	<code>getYear() : Number</code>	指定された Date オブジェクトの年をローカル時間で返します。
	<code> setDate(date:Number) : Number</code>	指定された Date オブジェクトの日付をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code> setFullYear (year:Number, [month:Number], [date:Number]) : Number</code>	指定された Date オブジェクトの年をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code> setHours (hour:Number) : Number</code>	指定された Date オブジェクトの時をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code> setMilliseconds (millisecond:Number) : Number</code>	指定された Date オブジェクトのミリ秒をローカル時間で設定し、新しい時間をミリ秒で返します。

オプション	シグネチャ	説明
	<code>setMinutes(minute: Number) : Number</code>	指定された Date オブジェクトの分をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code>setMonth(month: Number, [date: Number]) : Number</code>	指定された Date オブジェクトの月をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code>setSeconds(second: Number) : Number</code>	指定された Date オブジェクトの秒をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code>setTime(millisecond: Number) : Number</code>	指定された Date オブジェクトの日付を 1970 年 1 月 1 日 0 時からのミリ秒数で設定し、新しい時間をミリ秒で返します。
	<code>setUTCDate(date: Number) : Number</code>	指定された Date オブジェクトの日付を世界時で設定し、新しい時間をミリ秒で返します。
	<code>setUTCFullYear(year: Number, [month: Number], [date: Number]) : Number</code>	指定された Date オブジェクト (<i>my_date</i>) の年を世界時で設定し、新しい時間をミリ秒で返します。
	<code>setUTCHours(hour: Number, [minute: Number], [second: Number], [millisecond: Number]) : Number</code>	指定された Date オブジェクトの時を世界時で設定し、新しい時間をミリ秒で返します。
	<code>setUTCMilliseconds(millisecond: Number) : Number</code>	指定された Date オブジェクトのミリ秒を世界時で設定し、新しい時間をミリ秒で返します。
	<code>setUTCMinutes(minute: Number, [second: Number], [millisecond: Number]) : Number</code>	指定された Date オブジェクトの分を世界時で設定し、新しい時間をミリ秒で返します。
	<code>setUTCMonth(month: Number, [date: Number]) : Number</code>	指定された Date オブジェクトの月を世界時で設定し(必要であれば日付も設定可能)、新しい時間をミリ秒で返します。
	<code>setUTCSeconds(second: Number, [millisecond: Number]) : Number</code>	指定された Date オブジェクトの秒を世界時で設定し、新しい時間をミリ秒で返します。

オプション	シグネチャ	説明
	<code>setYear(year:Number) : Number</code>	指定された Date オブジェクトの年をローカル時間で設定し、新しい時間をミリ秒で返します。
	<code>toString() : String</code>	指定された Date オブジェクトのストリング値を読み取り可能な形式で返します。
static	<code>UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number]) : Number</code>	1970 年 1 月 1 日 0 時(世界時)からパラメータで指定された時刻までのミリ秒数を返します。
	<code>valueOf() : Number</code>	この Date に関して、1970 年 1 月 1 日 0 時(世界時)からのミリ秒数を返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty
(Object.hasOwnProperty メソッド), isPrototypeOf
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド),
registerClass (Object.registerClass メソッド), toString
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

Date コンストラクタ

```
public Date([yearOrTimevalue:Number], [month:Number], [date:Number],
[hour:Number], [minute:Number], [second:Number], [millisecond:Number])
```

指定された日時を保持する新しい Date オブジェクトを作成します。

Date() コンストラクタは、日付と、ミリ秒までの時刻を指定するために、最大 7 つまでパラメータ (year, month, ...、millisecond) を取ります。別の方針として、1970 年 1 月 1 日 0:00:00(世界時) を基準とする経過時間をミリ秒単位で表す 1 つの値を Date() コンストラクタに渡すこともできます。また、パラメータをまったく指定しないこともできます。その場合は、現在の日時が代入された Date() オブジェクトが作成されます。

Date オブジェクトの作成方法をいくつか次に示します。

```
var d1:Date = new Date();
var d3:Date = new Date(2000, 0, 1);
var d4:Date = new Date(65, 2, 6, 9, 30, 15, 0);
var d5:Date = new Date(-14159025000);
```

先頭行のコードでは、代入ステートメントが実行されたときの時刻が Date オブジェクトに設定されます。

2行目では、year、month、date パラメータを使って Date オブジェクトを作成しています。その結果、2000 年 1 月 1 日 0:00:00(世界時)になります。

3行目では、year、month、date パラメータを使って Date オブジェクトを作成しています。その結果、1965 年 3 月 6 日 09:30:15(世界時)(+0 ミリ秒)になります。ここでは、年が 2 衔の整数で指定されているので 1965 年として解釈されます。

4行目では、パラメータを 1つだけ使用しています。このパラメータは 1970 年 1 月 1 日 0:00:00(世界時)を基準とする経過時間をミリ秒単位で表す時間値です。この値が負の場合は 1970 年 1 月 1 日 0:00:00(世界時)よりも前の時刻を表すので、この例では、1969 年 7 月 21 日 02:56:15(世界時)になります。

パラメータ

yearOrTimevalue: Number (オプション)- 他にもパラメータが指定されている場合、この数値は年(1965 など)を表します。他にパラメータが設定されていない場合は、時間値を表します。数値が年を表す場合、0 ~ 99 の値は 1900 ~ 1999 を表します。それ以外の場合は年を 4 衔で指定する必要があります。数値が時間値を表す(他にパラメータが指定されない)場合は、1970 年 1 月 1 日 0:00:00 を基準とする経過時間をミリ秒単位で表す値になります。負の値は 1970 年 1 月 1 日 0:00:00(世界時)よりも前の時刻を表し、正の値はそれより後の時刻を表します。

month: Number (オプション)- 0(1月) ~ 11(12月) の整数です。

date: Number (オプション)- 1 ~ 31 の整数です。

hour: Number (オプション)- 0(0 時) ~ 23(午後 11 時) の整数です。

minute: Number (オプション)- 0 ~ 59 の整数です。

second: Number (オプション)- 0 ~ 59 の整数です。

millisecond: Number (オプション)- ミリ秒を表す 0 ~ 999 の整数です。

例

次の例では、現在の日時を調べます。

```
var now_date:Date = new Date();
```

次の例では、Mary の誕生日である 1974 年 8 月 12 日を表す新しい Date オブジェクトを作成します。月のパラメータはゼロから始まるので、この例では 8 ではなく 7 を使用します。

```
var maryBirthday:Date = new Date (74, 7, 12);
```

次の例では、新しい Date オブジェクトを作成し、Date.getMonth()、Date.getDate()、および Date.getFullYear() の戻り値を連結します。

```
var today_date:Date = new Date();
var date_str:String = ((today_date.getMonth()+1) + "/" + today_date.getDate() + /
    "+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

関連項目

[getMonth \(Date.getMonth メソッド\)](#), [getDate \(Date.getDate メソッド\)](#), [getFullYear \(Date.getFullYear メソッド\)](#)

getDate (Date.getDate メソッド)

public getDate() : Number

指定された Date オブジェクトの日付(1～31の整数)をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、Date.getMonth()、Date.getDate()、および Date.getFullYear() の戻り値を連結します。

```
var today_date:Date = new Date();
var date_str:String = (today_date.getDate() + "/" + (today_date.getMonth() + 1) + /
    "+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

関連項目

[getMonth \(Date.getMonth メソッド\)](#), [getFullYear \(Date.getFullYear メソッド\)](#)

getDay (Date.getDay メソッド)

public getDay() : Number

指定された Date オブジェクトの曜日(日曜日は0、月曜日は1など)をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 曜日を表す整数。

例

次の例では、新しい Date オブジェクトを作成し、getDay() を使用して現在の曜日を取得します。

```
var dayOfWeek_array:Array = new Array("Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday", "Saturday");
var today_date:Date = new Date();
var day_str:String = dayOfWeek_array[today_date.getDay()];
trace("Today is "+day_str);
```

getFullYear (Date.getFullYear メソッド)

public getFullYear():Number

指定された Date オブジェクトの年(2000など、4桁の数字)をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 年を表す整数。

例

次の例では、コンストラクタを使って Date オブジェクトを作成します。trace ステートメントは、getFullYear() メソッドの戻り値を表示します。

```
var my_date:Date = new Date();
trace(my_date.getFullYear()); // displays 104
trace(my_date.getFullYear()); // displays current year
```

getHours (Date.getHours メソッド)

public getHours():Number

指定された Date オブジェクトの時(0～23の整数)をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、getHours() メソッドを使ってオブジェクトの時の値を表示します。

```
var my_date:Date = new Date();
trace(my_date.getHours());

var my_date:Date = new Date();
```

```
var hourObj:Object = getHoursAmPm(my_date.getHours());
trace(hourObj.hours);
trace(hourObj.ampm);

function getHoursAmPm(hour24:Number):Object {
    var returnObj:Object = new Object();
    returnObj.ampm = (hour24<12) ? "AM" : "PM";
    var hour12:Number = hour24%12;
    if (hour12 == 0) {
        hour12 = 12;
    }
    returnObj.hours = hour12;
    return returnObj;
}
```

getLocaleLongDate (Date.getLocaleLongDate メソッド)

public getLocaleLongDate() : String

現在定義されているロケールに従ってフォーマットされた、長い形式の現在日付を表すストリングを返します。

メモ : 日付の形式は、モバイルデバイスおよびロケールによって異なります。

戻り値

String - 現在定義されているロケールに従ってフォーマットされた、長い形式の現在日付を表すストリング。

例

次の例では、コンストラクタを使って現在時に基づいて Date オブジェクトを作成します。また、次のように、getLocaleLongDate() メソッドを使用して、現在定義されているロケールに従ってフォーマットされた、長い形式の現在日付を返します。

```
var my_date:Date = new Date();
trace(my_date.getLocaleLongDate());
getLocaleLongDate() が返す戻り値のサンプルを以下に示します。
```

```
October 16, 2005
16 October 2005
```

getLocaleShortDate (Date.getLocaleShortDate メソッド)

public getLocaleShortDate() : String

現在定義されているロケールに従ってフォーマットされた、短い形式の現在日付を表すストリングを返します。

メモ : 日付の形式は、モバイルデバイスおよびロケールによって異なります。

戻り値

String - 現在定義されているロケールに従ってフォーマットされた、短い形式の現在日付を表すストリング。

例

次の例では、コンストラクタを使って現在時に基づいて Date オブジェクトを作成します。また、次のように、getLocaleShortDate() メソッドを使用して、現在定義されているロケールに従ってフォーマットされた、短い形式の現在日付を返します。

```
var my_date:Date = new Date();
trace(my_date.getLocaleShortDate());
```

getLocaleLongDate() が返す戻り値のサンプルを以下に示します。

```
10/16/2005
16-10-2005
```

getLocaleTime (Date.getLocaleTime メソッド)

public getLocaleTime() : String

現在定義されているロケールに従ってフォーマットされた現在時刻を表すストリングを返します。

メモ : 日付の形式は、モバイルデバイスおよびロケールによって異なります。

戻り値

String - 現在定義されているロケールに従ってフォーマットされた現在時刻を表すストリング。

例

次の例では、コンストラクタを使って現在時に基づいて Date オブジェクトを作成します。また、次のように、getLocaleTime() メソッドを使用して、現在のロケールの時刻を返します。

```
var my_date:Date = new Date();
trace(my_date.getLocaleTime());
```

getLocaleTime() が返す戻り値のサンプルを以下に示します。

```
6:10:44 PM
18:10:44
```

getMilliseconds (Date.getMilliseconds メソッド)

public getMilliseconds() : Number

指定された Date オブジェクトのミリ秒 (0 ~ 999 の整数) をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、getMilliseconds() メソッドを使ってオブジェクトのミリ秒の値を返します。

```
var my_date:Date = new Date();
trace(my_date.getMilliseconds());
```

getMinutes (Date.getMinutes メソッド)

public getMinutes() : Number

指定された Date オブジェクトの分 (0 ~ 59 の整数) をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、getMinutes() メソッドを使ってオブジェクトの分の値を返します。

```
var my_date:Date = new Date();
trace(my_date.getMinutes());
```

getMonth (Date.getMonth メソッド)

public getMonth() : Number

指定された Date オブジェクトの月 (1月は 0、2 月は 1 など) をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、`getMonth()` メソッドを使ってオブジェクトの月の値を返します。

```
var my_date:Date = new Date();
trace(my_date.getMonth());
```

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、`getMonth()` メソッドを使って現在の月を数値として表示し、さらに月の名前を表示します。

```
var my_date:Date = new Date();
trace(my_date.getMonth());
trace(getMonthAsString(my_date.getMonth()));
function getMonthAsString(month:Number):String {
    var monthNames_array:Array = new Array("January", "February", "March",
    "April", "May", "June", "July", "August", "September", "October", "November",
    "December");
    return monthNames_array[month];
}
```

getSeconds (Date.getSeconds メソッド)

public `getSeconds()` : Number

指定された Date オブジェクトの秒 (0 ~ 59 の整数) をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、`getSeconds()` メソッドを使ってオブジェクトの秒の値を返します。

```
var my_date:Date = new Date();
trace(my_date.getSeconds());
```

getTime (Date.getTime メソッド)

public `getTime()` : Number

指定された Date オブジェクトについて、1970 年 1 月 1 日 0 時 (世界時) からのミリ秒数を返します。このメソッドは、複数の Date オブジェクトを比較する際に特定の時間を表すのに使用します。

戻り値

Number - 整数。

例

次の例では、コンストラクタを使って現在の時刻に基づいて Date オブジェクトを作成し、`getTime()` メソッドを使って 1970 年 1 月 1 日 0 時からのミリ秒数を返します。

```
var my_date:Date = new Date();
trace(my_date.getTime());
```

getTimezoneOffset (Date.getTimezoneOffset メソッド)

public `getTimezoneOffset()` : Number

コンピュータのローカル時間と世界時の差(分単位)を返します。

戻り値

Number - 整数。

例

次の例では、サンフランシスコの現地夏時間と世界時の差を返します。夏時間は、Date オブジェクトで定義された日付が夏時間の期間内である場合だけ、返される結果に反映されます。この例の出力は 420 分であり、[出力] パネルに表示されます(7 時間 * 60 分 / 時 = 420 分)。この例は太平洋標準時の夏時間(PDT, GMT-0700)です。結果は地域と時期に応じて異なります。

```
var my_date:Date = new Date();
trace(my_date.getTimezoneOffset());
```

getUTCDate (Date.getUTCDate メソッド)

public `getUTCDate()` : Number

指定された Date オブジェクトの日付(1 ~ 31 の整数)を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、`Date.getUTCDate()` および `Date.getDate()` を使用します。`Date.getUTCDate()` の戻り値は、現地のタイムゾーンと世界時の関係によって、`Date.getDate()` の戻り値と異なる場合があります。

```
var my_date:Date = new Date(2004,8,25);
trace(my_date.getUTCDate()); // output: 25
```

関連項目

[getDate \(Date.getDate メソッド\)](#)

getUTCDay (Date.getUTCDay メソッド)

public getUTCDay() : Number

指定された Date オブジェクトの曜日 (日曜日は 0、月曜日は 1 など) を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、Date.getUTCDay() および Date.getDay() を使用します。Date.getUTCDay() の戻り値は、現地のタイムゾーンと世界時の関係によって、Date.getDay() の戻り値と異なる場合があります。

```
var today_date:Date = new Date();
trace(today_date.getDay()); // output will be based on local timezone
trace(today_date.getUTCDay()); // output will equal getDay() plus or minus one
```

関連項目

[getDay \(Date.getDay メソッド\)](#)

getUTCFullYear (Date.getUTCFullYear メソッド)

public getUTCFullYear() : Number

指定された Date オブジェクトの年 (4 衍表記) を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、Date.getUTCFullYear() および Date.getFullYear() を使用します。Date.getUTCFullYear() の戻り値は、今日の日付が 12 月 31 日または 1 月 1 日の場合に、現地のタイムゾーンと世界時の関係によって、Date.getFullYear() の戻り値と異なる場合があります。

```
var today_date:Date = new Date();
trace(today_date.getFullYear()); // display based on local timezone
trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

関連項目

[getFullYear \(Date.getFullYear メソッド\)](#)

getUTCHours (Date.getUTCHours メソッド)

public getUTCHours() : Number

指定された Date オブジェクトの時(0 ~ 23 の整数)を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、Date.getUTCHours() および Date.getHours() を使用します。Date.getUTCHours() の戻り値は、現地のタイムゾーンと世界時の関係によって、Date.getHours() の戻り値と異なる場合があります。

```
var today_date:Date = new Date();
trace(today_date.getHours()); // display based on local timezone
trace(today_date.getUTCHours()); // display equals getHours() plus or minus 12
```

関連項目

[getHours \(Date.getHours メソッド \)](#)

getUTCMilliseconds (Date.getUTCMilliseconds メソッド)

public getUTCMilliseconds() : Number

指定された Date オブジェクトのミリ秒(0 ~ 999 の整数)を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、getUTCMilliseconds() を使って Date オブジェクトのミリ秒の値を返します。

```
var today_date:Date = new Date();
trace(today_date.getUTCMilliseconds());
```

getUTCMMinutes (Date.getUTCMMinutes メソッド)

public getUTCMMinutes() : Number

指定された Date オブジェクトの分 (0 ~ 59 の整数) を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、getUTCMMinutes() を使って Date オブジェクトの分の値を返します。

```
var today_date:Date = new Date();
trace(today_date.getUTCMMinutes());
```

getUTCMonth (Date.getUTCMonth メソッド)

public getUTCMonth() : Number

指定された Date オブジェクトの月 (0 [1月] ~ 11 [12月]) を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、Date.getUTCMonth() および Date.getMonth() を使用します。Date.getUTCMonth() の戻り値は、今日の日付が月の最初または最後の日の場合に、現地のタイムゾーンと世界時の関係によって、Date.getMonth() の戻り値と異なる場合があります。

```
var today_date:Date = new Date();
trace(today_date.getMonth()); // output based on local timezone
trace(today_date.getUTCMonth()); // output equals getMonth() plus or minus 1
```

関連項目

[getMonth \(Date.getMonth メソッド\)](#)

getUTCSeconds (Date.getUTCSeconds メソッド)

public getUTCSeconds() : Number

指定された Date オブジェクトの秒(0 ~ 59 の整数)を世界時で返します。

戻り値

Number - 整数。

例

次の例では、新しい Date オブジェクトを作成し、getUTCSeconds() を使って Date オブジェクトの秒の値を返します。

```
var today_date:Date = new Date();
trace(today_date.getUTCSeconds());
```

getUTCYear (Date.getUTCYear メソッド)

public getUTCYear() : Number

この Date の年を世界時(UTC)で返します。年は、4桁の年から 1900 を引いて示されます。たとえば、2000 年は 100 と表されます。

戻り値

Number -

例

次の例では、新しい Date オブジェクトを作成し、Date.getUTCFullYear() および Date.getFullYear() を使用します。Date.getUTCFullYear() の戻り値は、今日の日付が 12 月 31 日または 1 月 1 日の場合に、現地のタイムゾーンと世界時の関係によって、Date.getFullYear() の戻り値と異なる場合があります。

```
var today_date:Date = new Date();

trace(today_date.getFullYear()); // display based on local timezone

trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

getYear (Date.getYear メソッド)

public getYear() : Number

指定された Date オブジェクトの年をローカル時間で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。年は、4 衝の年から 1900 を引いて示されます。たとえば、2000 年は 100 と表されます。

戻り値

Number - 整数。

例

次の例では、月および年が 2004 年 5 月に設定された新しい Date オブジェクトを作成します。この場合、Date.getYear() メソッドは 104 を返し、Date.getFullYear() メソッドは 2004 を返します。

```
var today_date:Date = new Date(2004,4);
trace(today_date.getYear()); // output: 104
trace(today_date.getFullYear()); // output: 2004
```

関連項目

[getFullYear \(Date.getFullYear メソッド \)](#)

setDate (Date.setDate メソッド)

public setDate(date:Number) : Number

指定された Date オブジェクトの日付をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

date:Number - 1～31 の整数です。

戻り値

Number - 整数。

例

次の例では、日付を 2004 年 5 月 15 日に設定した新しい Date オブジェクトを作成した後、Date.setDate() を使って日付を 2004 年 5 月 25 日に変更します。

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getDate()); //displays 15
today_date.setDate(25);
trace(today_date.getDate()); //displays 25
```

setFullYear (Date.setFullYear メソッド)

```
public setFullYear(year:Number, [month:Number], [date:Number]): Number
```

指定された Date オブジェクトの年をローカル時間で設定し、新しい時間をミリ秒で返します。month パラメータと date パラメータを指定すると、両方は同時にローカル時間に設定されます。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

このメソッドを呼び出しても、指定した Date オブジェクトの他のフィールドは変更されません。ただし、このメソッドを呼び出した結果として曜日が変わった場合には、Date.getUTCDay() と Date.getDay() は新しい値を返すことがあります。

パラメータ

year: Number - 年を指定する 4 衔の数値です。2 衔の数値は 4 衔の年の省略形を表しません。たとえば、99 は 1999 年ではなく、99 年です。

month: Number (オプション) - 0(1月)～11(12月) の整数です。このパラメータを省略した場合、指定した Date オブジェクトの month フィールドは変更されません。

date: Number (オプション) - 1～31 の数値です。このパラメータを省略した場合、指定した Date オブジェクトの date フィールドは変更されません。

戻り値

Number - 整数。

例

次の例では、日付を 2004 年 5 月 15 日に設定した新しい Date オブジェクトを作成した後、Date.setFullYear() を使って日付を 2002 年 5 月 15 日に変更します。

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getFullYear()); //output: 2004
my_date.setFullYear(2002);
trace(my_date.getFullYear()); //output: 2002
```

関連項目

[getUTCDay \(Date.getUTCDay メソッド\)](#), [getDay \(Date.getDay メソッド\)](#)

setHours (Date.setHours メソッド)

public setHours(hour:Number) : Number

指定された Date オブジェクトの時をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

hour:Number 0(0 時)～23(午後 11 時)の整数です。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00 に設定した新しい Date オブジェクトを作成した後、Date.setHours() を使って時刻を午後 4:00 に変更します。

```
var my_date:Date = new Date(2004,4,15,8);
trace(my_date.getHours()); // output: 8
my_date.setHours(16);
trace(my_date.getHours()); // output: 16
```

setMilliseconds (Date.setMilliseconds メソッド)

public setMilliseconds(millisecond:Number) : Number

指定された Date オブジェクトのミリ秒をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

millisecond:Number - 0～999 の整数です。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:30 分 250 ミリ秒に設定した新しい Date オブジェクトを作成した後、Date.setMilliseconds() を使ってミリ秒の値を 575 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getMilliseconds()); // output: 250
my_date.setMilliseconds(575);
trace(my_date.getMilliseconds()); // output: 575
```

setMinutes (Date.setMinutes メソッド)

public setMinutes(minute:Number) : Number

指定された Date オブジェクトの分をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

minute:Number - 0 ~ 59 の整数です。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00 に設定した新しい Date オブジェクトを作成した後、Date.setMinutes() を使って時刻を午前 8:30 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getMinutes()); // output: 0
my_date.setMinutes(30);
trace(my_date.getMinutes()); // output: 30
```

setMonth (Date.setMonth メソッド)

public setMonth(month:Number, [date:Number]) : Number

指定された Date オブジェクトの月をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

month:Number - 0(1月) ~ 11(12月) の整数です。

date:Number (オプション) - 1 ~ 31 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの date フィールドは変更されません。

戻り値

Number - 整数。

例

次の例では、日付を 2004 年 5 月 15 日に設定した新しい Date オブジェクトを作成した後、Date.setMonth() を使って日付を 2004 年 6 月 15 日に変更します。

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getMonth()); //output: 4
my_date.setMonth(5);
trace(my_date.getMonth()); //output: 5
```

setSeconds (Date.setSeconds メソッド)

public setSeconds(second:Number) : Number

指定された Date オブジェクトの秒をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

second:Number - 0 ~ 59 の整数です。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00:00 に設定した新しい Date オブジェクトを作成した後、Date.setSeconds() を使って時刻を午前 8:00:45 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getSeconds()); // output: 0
my_date.setSeconds(45);
trace(my_date.getSeconds()); // output: 45
```

setTime (Date.setTime メソッド)

public setTime(millisecond:Number) : Number

指定された Date オブジェクトの日付を 1970 年 1 月 1 日 0 時からのミリ秒数で設定し、新しい時間をミリ秒で返します。

パラメータ

millisecond:Number - 数値。1 月 1 日 0 時 (世界時) を 0 とする整数値です。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00 に設定した新しい Date オブジェクトを作成した後、Date.setTime() を使って時刻を午前 8:30 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,0,0);
var myDate_num:Number = my_date.getTime(); // convert my_date to milliseconds
myDate_num += 30 * 60 * 1000; // add 30 minutes in milliseconds
my_date.setTime(myDate_num); // set my_date Date object 30 minutes forward
trace(my_date.getFullYear()); // output: 2004
trace(my_date.getMonth()); // output: 4
```

```
trace(my_date.getDate()); // output: 15  
trace(my_date.getHours()); // output: 8  
trace(my_date.getMinutes()); // output: 30
```

setUTCDate (Date.setUTCDate メソッド)

public setUTCDate(date:Number) : Number

指定された Date オブジェクトの日付を世界時で設定し、新しい時間をミリ秒で返します。このメソッドを呼び出しても、指定した Date オブジェクトの他のフィールドは変更されません。ただし、このメソッドを呼び出した結果として曜日が変わった場合には、Date.getUTCDay() と Date.getDay() は新しい値を返すことがあります。

パラメータ

date:Number - 数値。1～31の整数です。

戻り値

Number - 整数。

例

次の例では、今日の日付に設定した新しい Date オブジェクトを作成した後、Date.setUTCDate() を使って日にちの値を 10 に変更し、さらに 25 に変更します。

```
var my_date:Date = new Date();  
my_date.setUTCDate(10);  
trace(my_date.getUTCDate()); // output: 10  
my_date.setUTCDate(25);  
trace(my_date.getUTCDate()); // output: 25
```

関連項目

[getUTCDay \(Date.getUTCDay メソッド\)](#), [getDay \(Date.getDay メソッド\)](#)

setUTCFullYear (Date.setUTCFullYear メソッド)

public setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number

指定された Date オブジェクト (*my_date*) の年を世界時で設定し、新しい時間をミリ秒で返します。オプションとして、このメソッドは指定された Date オブジェクトが表す月日も設定できます。このメソッドを呼び出しても、指定した Date オブジェクトの他のフィールドは変更されません。ただし、このメソッドを呼び出した結果として曜日が変わった場合には、Date.getUTCDay() と Date.getDay() は新しい値を返すことがあります。

パラメータ

`year:Number` - 4桁の年(2000など)を表す整数です。

`month:Number` (オプション) - 0(1月)～11(12月)の整数です。このパラメータを省略した場合、指定した Date オブジェクトの month フィールドは変更されません。

`date:Number` (オプション) - 1～31の整数です。このパラメータを省略した場合、指定した Date オブジェクトの date フィールドは変更されません。

戻り値

`Number` - 整数。

例

次の例では、今日の日付に設定した新しい Date オブジェクトを作成した後、`Date.setUTCFullYear()` を使って年の値を 2001 に変更し、さらに日付を 1995 年 5 月 25 日に変更します。

```
var my_date:Date = new Date();
my_date.setUTCFullYear(2001);
trace(my_date.getUTCFullYear()); // output: 2001
my_date.setUTCFullYear(1995, 4, 25);
trace(my_date.getUTCFullYear()); // output: 1995
trace(my_date.getUTCMonth()); // output: 4
trace(my_date.getUTCDate()); // output: 25
```

関連項目

[getUTCDay \(Date.getUTCDay メソッド\)](#), [getDay \(Date.getDay メソッド\)](#)

setUTCHours (Date.setUTCHours メソッド)

```
public setUTCHours(hour:Number, [minute:Number], [second:Number],
    [millisecond:Number]):Number
```

指定された Date オブジェクトの時を世界時で設定し、新しい時間をミリ秒で返します。

パラメータ

`hour:Number` - 数値。0(0時)～23(午後11時)の整数です。

`minute:Number` (オプション) - 数値。0～59 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの minutes フィールドは変更されません。

`second:Number` (オプション) - 数値。0～59 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの seconds フィールドは変更されません。

`millisecond:Number` (オプション) - 数値。0～999 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの milliseconds フィールドは変更されません。

戻り値

[Number](#) - 整数。

例

次の例では、今日の日付に設定した新しい Date オブジェクトを作成した後、`Date.setUTCHours()` を使って時刻を午前 8:30 に変更し、さらに午後 5:30:47 に変更します。

```
var my_date:Date = new Date();
my_date.setUTCHours(8,30);
trace(my_date.getUTCHours()); // output: 8
trace(my_date.getUTCMinutes()); // output: 30
my_date.setUTCHours(17,30,47);
trace(my_date.getUTCHours()); // output: 17
trace(my_date.getUTCMinutes()); // output: 30
trace(my_date.getUTCSeconds()); // output: 47
```

setUTCMilliseconds (Date.setUTCMilliseconds メソッド)

`public setUTCMilliseconds(millisecond:Number) : Number`

指定された Date オブジェクトのミリ秒を世界時で設定し、新しい時間をミリ秒で返します。

パラメータ

`millisecond:Number` - 0 ~ 999 の整数です。

戻り値

[Number](#) - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:30 分 250 ミリ秒に設定した新しい Date オブジェクトを作成した後、`Date.setUTCMilliseconds()` を使ってミリ秒の値を 575 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getUTCMilliseconds()); // output: 250
my_date.setUTCMilliseconds(575);
trace(my_date.getUTCMilliseconds()); // output: 575
```

setUTCMMinutes (Date.setUTCMMinutes メソッド)

```
public setUTCMMinutes(minute:Number, [second:Number], [millisecond:Number]) : Number
```

指定された Date オブジェクトの分を世界時で設定し、新しい時間をミリ秒で返します。

パラメータ

minute:Number - 0 ~ 59 の整数です。

second:Number (オプション) - 0 ~ 59 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの seconds フィールドは変更されません。

millisecond:Number (オプション) - 0 ~ 999 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの milliseconds フィールドは変更されません。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00 に設定した新しい Date オブジェクトを作成した後、Date.setUTCMMinutes() を使って時刻を午前 8:30 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getUTCMMinutes()); // output: 0
my_date.setUTCMMinutes(30);
trace(my_date.getUTCMMinutes()); // output: 30
```

setUTCMonth (Date.setUTCMonth メソッド)

```
public setUTCMonth(month:Number, [date:Number]) : Number
```

指定された Date オブジェクトの月を世界時で設定し（必要であれば日付も設定可能）、新しい時間をミリ秒で返します。このメソッドを呼び出しても、指定した Date オブジェクトの他のフィールドは変更されません。ただし、date パラメータに値を指定した結果として曜日が変わった場合は、Date.getUTCDay() と Date.getDay() は新しい値を返すことがあります。

パラメータ

month:Number - 0 (1月) ~ 11 (12月) の整数です。

date:Number (オプション) - 1 ~ 31 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの date フィールドは変更されません。

戻り値

Number - 整数。

例

次の例では、日付を 2004 年 5 月 15 日に設定した新しい Date オブジェクトを作成した後、
Date.setMonth() を使って日付を 2004 年 6 月 15 日に変更します。

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getUTCMonth()); // output: 4
today_date.setUTCMonth(5);
trace(today_date.getUTCMonth()); // output: 5
```

関連項目

[getUTCDay \(Date.getUTCDay メソッド\)](#), [getDay \(Date.getDay メソッド\)](#)

setUTCSeconds (Date.setUTCSeconds メソッド)

public setUTCSeconds(second:Number, [millisecond:Number]): Number

指定された Date オブジェクトの秒を世界時で設定し、新しい時間をミリ秒で返します。

パラメータ

second:Number - 0 ~ 59 の整数です。

millisecond:Number (オプション) - 0 ~ 999 の整数です。このパラメータを省略した場合、指定した Date オブジェクトの milliseconds フィールドは変更されません。

戻り値

Number - 整数。

例

次の例では、日時を 2004 年 5 月 15 日午前 8:00:00 に設定した新しい Date オブジェクトを作成した後、Date.setSeconds() を使って時刻を午前 8:30:45 に変更します。

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getUTCSeconds()); // output: 0
my_date.setUTCSeconds(45);
trace(my_date.getUTCSeconds()); // output: 45
```

setYear (Date.setYear メソッド)

public setYear(year:Number) : Number

指定された Date オブジェクトの年をローカル時間で設定し、新しい時間をミリ秒で返します。ローカル時間は、Flash Player を実行しているオペレーティングシステムによって決まります。

パラメータ

year:Number - 年を表す数値です。year が 0 ~ 99 の整数である場合、setYear は、1900 + year の値を年として設定します。それ以外の場合は、year パラメータの値を年として設定します。

戻り値

Number - 整数。

例

次の例では、日付を 2004 年 5 月 25 日に設定した新しい Date オブジェクトを作成した後、setYear() を使って年の値を 1999 に変更し、さらに 2003 に変更します。

```
var my_date:Date = new Date(2004,4,25);
trace(my_date.getYear()); // output: 104
trace(my_date.getFullYear()); // output: 2004
my_date.setYear(99);
trace(my_date.getYear()); // output: 99
trace(my_date.getFullYear()); // output: 1999
my_date.setYear(2003);
trace(my_date.getYear()); // output: 103
trace(my_date.getFullYear()); // output: 2003
```

toString (Date.toString メソッド)

public toString() : String

指定された Date オブジェクトのストリング値を読み取り可能な形式で返します。

戻り値

String - ストリング。

例

次の例では、dateOfBirth_date という Date オブジェクト内の情報をストリングとして返します。trace ステートメントからの出力はローカル時間に基づいて変化します。太平洋標準時の夏時間の場合、出力は世界時よりも 7 時間前の値となる、Mon Aug 12 18:15:00 GMT-0700 1974 になります。

```
var dateOfBirth_date:Date = new Date(74, 7, 12, 18, 15);
trace (dateOfBirth_date);
trace (dateOfBirth_date.toString());
```

UTC (Date.UTC メソッド)

```
public static UTC(year:Number, month:Number, [date:Number], [hour:Number],  
[minute:Number], [second:Number], [millisecond:Number]) : Number
```

1970年1月1日0時(世界時)からパラメータで指定された時刻までのミリ秒数を返します。このメソッドは、特定の Date オブジェクトからでなく Date クラスから呼び出される静的なメソッドです。このメソッドを使用すると、世界時を採用する Date オブジェクトを作成できます。一方、Date コンストラクタはローカル時間を採用します。

パラメータ

year: Number - 年を表す 4 桁の数値(2000 など)です。

month: Number - 0(1月)～11(12月)の整数です。

date: Number (オプション)-1～31の整数です。

hour: Number (オプション)-0(0時)～23(午後11時)の整数です。

minute: Number (オプション)-0～59 の整数です。

second: Number (オプション)-0～59 の整数です。

millisecond: Number (オプション)-0～999 の整数です。

戻り値

Number - 整数。

例

次の例では、世界時で定義された新しい Date オブジェクト maryBirthday_date を作成します。これは、コンストラクタメソッド new Date を使用して、世界時に対応した Date オブジェクトを作成する例です。出力は、ローカル時間に基づいて変化します。太平洋標準時の夏時間の場合、出力は世界時よりも 7 時間前の値となる、Sun Aug 11 17:00:00 GMT-0700 1974 になります。

```
var maryBirthday_date:Date = new Date(Date.UTC(1974, 7, 12));  
trace(maryBirthday_date);
```

valueOf (Date.valueOf メソッド)

```
public valueOf() : Number
```

この Date に関して、1970年1月1日0時(世界時)からのミリ秒数を返します。

戻り値

Number - ミリ秒数。

Error

```
Object
 |
 +-Error
```

```
public class Error
extends Object
```

スクリプト内で発生したエラーについての情報を含みます。Error オブジェクトを作成するには、Error コンストラクタ関数を使用します。一般には、try コードブロック内から新しい Error オブジェクトの throw を実行します。そして、catch コードブロックまたは finally コードブロックでこれをキャッチします。

Error クラスのサブクラスを作成し、そのサブクラスのインスタンスをスローすることもできます。

プロパティ一覧

オプション	プロパティ	説明
	<code>message:String</code>	Error オブジェクトに関連付けられたメッセージです。
	<code>name:String</code>	Error オブジェクトの名前です。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>Error ([message:String])</code>	新しい Error オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>toString() : String</code>	デフォルトでは "Error" というストリングを返します。 Error.message が定義されている場合は、その値を返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

Error コンストラクタ

public Error([message:String])

新しい Error オブジェクトを作成します。message を指定した場合は、その値がオブジェクトの Error.message プロパティに割り当てられます。

パラメータ

message:String(オプション)-Error オブジェクトに関連付けられたストリング。

例

次の例の関数は、受け取った 2 つのストリングが同じでない場合に、指定のメッセージを使用してエラーをスローします。

```
function compareStrings(str1_str:String, str2_str:String):Void {  
    if (str1_str != str2_str) {  
        throw new Error("Strings do not match.");  
    }  
}  
try {  
    compareStrings("Dog", "dog");  
    // output: Strings do not match.  
} catch (e_err:Error) {  
    trace(e_err.toString());  
}
```

関連項目

[throw ステートメント](#), [try..catch..finally ステートメント](#)

message (Error.message プロパティ)

public message : [String](#)

Error オブジェクトに関連付けられたメッセージです。デフォルトでは、このプロパティの値は "Error" です。Error オブジェクトを作成する際に、Error コンストラクタ関数にエラーストリングを渡すことで message プロパティを指定できます。

例

次の例の関数は、theNum に入力されたパラメータに応じて、指定されたメッセージをスローします。2つの数値を使って除算を行える場合は、SUCCESS というストリングとその結果が表示されます。0 による除算を行おうとした場合、または1つしかパラメータが入力されていない場合は、それに対応するエラーが表示されます。

```
function divideNum(num1:Number, num2:Number):Number {
    if (isNaN(num1) || isNaN(num2)) {
        throw new Error("divideNum function requires two numeric parameters.");
    } else if (num2 == 0) {
        throw new Error("cannot divide by zero.");
    }
    return num1/num2;
}
try {
    var theNum:Number = divideNum(1, 0);
    trace("SUCCESS! "+theNum);
} catch (e_err:Error) {
    trace("ERROR! "+e_err.message);
    trace("\t"+e_err.name);
}
```

この ActionScript の除算する数値を変えずにテストすると、0 による除算を行うことになり、[出力] パネルにエラーが表示されます。

関連項目

[throw ステートメント](#) , [try..catch..finally ステートメント](#)

name (Error.name プロパティ)

public name : String

Error オブジェクトの名前です。デフォルトでは、このプロパティの値は "Error" です。

例

次の例の関数は、除算する 2 つの数値に応じて、指定されたエラーをスローします。タイムラインのフレーム 1 に次の ActionScript を追加します。

```
function divideNumber(numerator:Number, denominator:Number):Number {
    if (isNaN(numerator) || isNaN(denominator)) {
        throw new Error("divideNum function requires two numeric parameters.");
    } else if (denominator == 0) {
        throw new DivideByZeroError();
    }
    return numerator/denominator;
}
try {
    var theNum:Number = divideNumber(1, 0);
    trace("SUCCESS! "+theNum);
    // output: DivideByZeroError -> Unable to divide by zero.
} catch (e_err:DivideByZeroError) {
    // divide by zero error occurred
    trace(e_err.name+" -> "+e_err.toString());
} catch (e_err:Error) {
    // generic error occurred
    trace(e_err.name+" -> "+e_err.toString());
}
```

カスタムエラーを追加するには、次のコードを "DivideByZeroError.as" ファイルに追加し、FLA ドキュメントと同じディレクトリにクラスファイルを保存します。

```
class DivideByZeroError extends Error {
    var name:String = "DivideByZeroError";
    var message:String = "Unable to divide by zero.";
}
```

関連項目

[throw ステートメント](#), [try..catch..finally ステートメント](#)

toString (Error.toString メソッド)

```
public toString() : String
```

デフォルトでは "Error" というストリングを返します。Error.message が定義されている場合は、その値を返します。

戻り値

[String](#) - ストリング。

例

次の例の関数は、受け取った 2 つのストリングが同じでない場合に、指定のメッセージを使用してエラーをスローします。

```
function compareStrings(str1_str:String, str2_str:String):Void {
    if (str1_str != str2_str) {
        throw new Error("Strings do not match.");
    }
}
try {
    compareStrings("Dog", "dog");
    // output: Strings do not match.
} catch (e_err:Error) {
    trace(e_err.toString());
}
```

関連項目

[message \(Error.message プロパティ\)](#), [throw ステートメント](#), [try..catch..finally ステートメント](#)

ExtendedKey

```
Object
|
+-ExtendedKey
```

```
public class ExtendedKey
extends Object
```

Key.getCode() メソッドで返すことができる拡張キーコードを提供します。

例

次の例では、キーが押されたときに呼び出されるリスナーを作成します。このリスナーは、Key.getCode() メソッドを使用して、押されたキーのキーコードを取得します。

```

var myListener = new Object();

myListener.onKeyDown = function() {
    var code = Key.getCode();
    switch(code) {
        case 50:
            trace("number 2 down");
            break;
        case Key.ENTER:
            trace("enter down");
            break;
        case ExtendedKey.SOFT1:
            trace("soft1 down");
            break;
        default:
            trace(code + " down");
            break;
    }
}

myListener.onKeyUp = function() {
    text2 = "onKeyUp called";
}

Key.addListener(myListener);

```

関連項目

[getCode \(Key.getCode メソッド\)](#)

プロパティ一覧

オプション	プロパティ	説明
static	SOFT1:String	SOFT1 ソフトキーのキーコード値。
static	SOFT10:String	SOFT10 ソフトキーのキーコード値。
static	SOFT11:String	SOFT11 ソフトキーのキーコード値。
static	SOFT12:String	SOFT12 ソフトキーのキーコード値。
static	SOFT2:String	SOFT2 ソフトキーのキーコード値。
static	SOFT3:String	SOFT3 ソフトキーのキーコード値。
static	SOFT4:String	SOFT4 ソフトキーのキーコード値。
static	SOFT5:String	SOFT5 ソフトキーのキーコード値。
static	SOFT6:String	SOFT6 ソフトキーのキーコード値。

オプション	プロパティ	説明
static	SOFT7:String	SOFT7 ソフトキーのキーコード値。
static	SOFT8:String	SOFT8 ソフトキーのキーコード値。
static	SOFT9:String	SOFT9 ソフトキーのキーコード値。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

メソッド一覧

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty (Object.hasOwnProperty メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf メソッド), registerClass (Object.registerClass メソッド), toString (Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf (Object.valueOf メソッド), watch (Object.watch メソッド)
```

SOFT1(ExtendedKey.SOFT1 プロパティ)

public static SOFT1 : String

SOFT1 ソフトキーのキーコード値。SOFT1 キーコードは常に左ソフトキーに対応し、SOFT2 は常に右ソフトキーに対応します。

例

次の例では、左ソフトキーと右ソフトキーを処理するリスナーを作成します。

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode();
    switch (keyCode) {
        case ExtendedKey.SOFT1:
            // Handle left soft key.
            break;
        case ExtendedKey.SOFT2:
            // Handle right soft key
            break;
    }
}
Key.addListener(myListener);
```

SOFT10 (ExtendedKey.SOFT10 プロパティ)

public static SOFT10 : [String](#)

SOFT10 ソフトキーのキーコード値。

SOFT11 (ExtendedKey.SOFT11 プロパティ)

public static SOFT11 : [String](#)

SOFT11 ソフトキーのキーコード値。

SOFT12 (ExtendedKey.SOFT12 プロパティ)

public static SOFT12 : [String](#)

SOFT12 ソフトキーのキーコード値。

SOFT2 (ExtendedKey.SOFT2 プロパティ)

public static SOFT2 : [String](#)

SOFT2 ソフトキーのキーコード値。SOFT2 キーコードは常に右ソフトキーに対応し、SOFT1 キーコードは常に左ソフトキーに対応します。

関連項目

[SOFT1 \(ExtendedKey.SOFT1 プロパティ \)](#)

SOFT3 (ExtendedKey.SOFT3 プロパティ)

public static SOFT3 : [String](#)

SOFT3 ソフトキーのキーコード値。

SOFT4 (ExtendedKey.SOFT4 プロパティ)

public static SOFT4 : [String](#)

SOFT4 ソフトキーのキーコード値。

SOFT5 (ExtendedKey.SOFT5 プロパティ)

public static SOFT5 : [String](#)

SOFT5 ソフトキーのキーコード値。

SOFT6 (ExtendedKey.SOFT6 プロパティ)

public static SOFT6 : `String`

SOFT6 ソフトキーのキーコード値。

SOFT7 (ExtendedKey.SOFT7 プロパティ)

public static SOFT7 : `String`

SOFT7 ソフトキーのキーコード値。

SOFT8 (ExtendedKey.SOFT8 プロパティ)

public static SOFT8 : `String`

SOFT8 ソフトキーのキーコード値。

SOFT9 (ExtendedKey.SOFT9 プロパティ)

public static SOFT9 : `String`

SOFT9 ソフトキーのキーコード値。

Function

```
Object  
|  
+-Function
```

public dynamic class Function
extends `Object`

ActionScript のユーザー定義関数とビルトイン関数は、どちらも Function クラスのインスタンスである Function オブジェクトで表されます。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

メソッド一覧

オプション	シグネチャ	説明
	<code>apply(thisObject:Object, [argArray:Array])</code>	ActionScript が呼び出す関数内で使用される thisObject の値を指定します。
	<code>call(thisObject:Object, [parameter1:Object])</code>	Function オブジェクトが表す関数を呼び出します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

apply (Function.apply メソッド)

`public apply(thisObject:Object, [argArray:Array])`

ActionScript が呼び出す関数内で使用される thisObject の値を指定します。このメソッドは、呼び出される関数に渡されるパラメータも指定します。apply() は Function クラスのメソッドなので、ActionScript 内のすべての Function オブジェクトのメソッドとしても使用できます。

パラメータは、カンマ区切りリストとしてパラメータを指定する Function.call() とは異なり、Array オブジェクトとして指定します。これは、スクリプトが実際に実行されるまで、渡されるパラメータ数が不明である場合にも便利です。

呼び出された関数が戻り値として指定する値を返します。

パラメータ

`thisObject:Object` - myFunction の適用先のオブジェクト。

`argArray:Array` (オプション) - エレメントをパラメータとして myFunction に渡す配列。

戻り値

呼び出された関数が指定する任意の値。

例

次の2つの関数呼び出しが同じです。

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

次の簡単な例は、apply()を使用してパラメータの配列を渡す方法を示しています。

```
function theFunction() {
    trace(arguments);
}

// create a new array to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
theFunction.apply(null,firstArray);
// outputs: 1,2,3
```

```
// create a second array to pass as a parameter to apply()
var secondArray:Array = new Array("a", "b", "c");
theFunction.apply(null,secondArray);
// outputs a,b,c
```

次の例では、apply()を使用してパラメータの配列を渡し、thisの値を指定します。

```
// define a function
function theFunction() {
    trace("this == myObj? " + (this == myObj));
    trace("arguments: " + arguments);
}

// instantiate an object
var myObj:Object = new Object();

// create arrays to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
var secondArray:Array = new Array("a", "b", "c");

// use apply() to set the value of this to be myObj and send firstArray
theFunction.apply(myObj,firstArray);
// output:
// this == myObj? true
// arguments: 1,2,3

// use apply() to set the value of this to be myObj and send secondArray
theFunction.apply(myObj,secondArray);
// output:
// this == myObj? true
// arguments: a,b,c
```

関連項目

[call \(Function.call メソッド\)](#)

call (Function.call メソッド)

```
public call(thisObject:Object, [parameter1:Object])
```

Function オブジェクトが表す関数を呼び出します。ActionScript のすべての関数は Function オブジェクトによって表されます。したがって、すべての関数は、このメソッドをサポートしています。

ほとんどの場合、このメソッドの代わりに関数呼び出し演算子 (()) を使用できます。関数呼び出し演算子を使うと、コードが簡潔になり読みやすくなります。このメソッドは、主に関数呼び出しの thisObject パラメータを明示的に制御する必要がある場合に役立ちます。通常、関数をオブジェクトのメソッドとして、関数の本体内で呼び出すと、次のように thisObject が myObject に設定されます。

```
myObject.myMethod(1, 2, 3);
```

thisObject が他の異なる場所をポイントするように設定する場合もあります。たとえば、オブジェクトのメソッドとして呼び出す関数が、実際には、そのオブジェクトのメソッドとして格納されていない場合などです。

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

関数をオブジェクトのメソッドとして呼び出さずに通常の関数として呼び出すには、thisObject パラメータに値 null を渡します。たとえば、次の 2 つの関数呼び出しは同じです。

```
Math.sin(Math.PI / 4)
```

```
Math.sin.call(null, Math.PI / 4)
```

呼び出された関数が戻り値として指定する値を返します。

パラメータ

thisObject:Object - 関数の本体内で thisObject の値を指定するオブジェクト。

parameter1:Object (オプション) - myFunction に渡すパラメータ。指定できるパラメータの数は 0 個以上です。

戻り値

例

次の例では、Function.call() メソッドを使用することで、関数をオブジェクトに格納しないまま、別のオブジェクトのメソッドとして動作させます。

```
function myObject() {  
}  
function myMethod(obj) {  
    trace("this == obj? " + (this == obj));  
}  
var obj:Object = new myObject();  
myMethod.call(obj, obj);
```

`trace()` ステートメントの出力は次のようにになります。

```
this == obj? true
```

関連項目

[apply \(Function.apply メソッド\)](#)

Key

`Object`



```
public class Key  
extends Object
```

`Key` クラスはトップレベルのクラスで、コンストラクタを実行しなくとも、そのメソッドやプロパティを使用できます。`Key` クラスのメソッドを使用すると、標準キーボードで制御できるインターフェイスを作成できます。`Key` クラスのプロパティは、矢印キー、`PageUp` キー、`PageDown` キーなど、アプリケーションの制御に最も一般的に使用されるキーを表す定数です。

関連項目

[ExtendedKey](#)

プロパティ一覧

オプション	プロパティ	説明
static	<code>BACKSPACE:Number</code>	<code>Backspace</code> キーのキーコード値(8)。
static	<code>CAPSLOCK:Number</code>	<code>CapsLock</code> キーのキーコード値(20)。
static	<code>CONTROL:Number</code>	<code>Control</code> キーのキーコード値(17)。
static	<code>DELETEKEY:Number</code>	<code>Delete</code> キーのキーコード値(46)。
static	<code>DOWN:Number</code>	下矢印キーのキーコード値(40)。
static	<code>END:Number</code>	<code>End</code> キーのキーコード値(35)。
static	<code>ENTER:Number</code>	<code>Enter</code> キーのキーコード値(13)。
static	<code>ESCAPE:Number</code>	<code>Escape</code> キーのキーコード値(27)。
static	<code>HOME:Number</code>	<code>Home</code> キーのキーコード値(36)。
static	<code>INSERT:Number</code>	<code>Insert</code> キーのキーコード値(45)。
static	<code>LEFT:Number</code>	左矢印キーのキーコード値(37)。

オプション	プロパティ	説明
static	<code>_listeners:Array</code> (読み取り専用)	Key オブジェクトに登録したすべてのリスナーオブジェクトへの参照のリスト。
static	<code>PGDN:Number</code>	PageDown キーのキーコード値(34)。
static	<code>PGUP:Number</code>	PageUp キーのキーコード値(33)。
static	<code>RIGHT:Number</code>	右矢印キーのキーコード値(39)。
static	<code>SHIFT:Number</code>	Shift キーのキーコード値(16)。
static	<code>SPACE:Number</code>	スペースバーのキーコード値(32)。
static	<code>TAB:Number</code>	Tab キーのキーコード値(9)。
static	<code>UP:Number</code>	上矢印キーのキーコード値(38)。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
<code>onKeyDown</code> = <code>function() {}</code>	キーを押すと通知されます。
<code>onKeyUp</code> = <code>function() {}</code>	キーを離すと通知されます。

メソッド一覧

オプション	シグネチャ	説明
static	<code>addListener(listener: Object) : Void</code>	onKeyDown 通知と onKeyUp 通知を受け取るオブジェクトを登録します。
static	<code>getAscii() : Number</code>	最後に押されたか離されたキーの ASCII コードを返します。
static	<code>getCode() : Number</code>	最後に押されたキーのキーコード値を返します。
static	<code>isDown(code:Number) : Boolean</code>	code に指定されたキーが押された場合は true、それ以外の場合は false を返します。
static	<code>removeListener (listener:Object) : Boolean</code>	Key.addListener() を使用して以前に登録したオブジェクトを削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (Key.addListener メソッド)

public static addListener(listener:[Object](#)) : Void

onKeyDown 通知と onKeyUp 通知を受け取るオブジェクトを登録します。キーを押すか離すと、入力フォーカスに関係なく、addListener() に登録されているすべてのリスナーオブジェクトの onKeyDown メソッドまたは onKeyUp メソッドが呼び出されます。複数のオブジェクトでキーボード通知を受け取ることができます。リスナーが登録済みの場合は何も変化しません。

パラメータ

listener:[Object](#) - onKeyDown メソッドと onKeyUp メソッドを持つオブジェクト。

例

次の例では、新しいリスナーオブジェクトを作成し、onKeyDown と onKeyUp の関数を定義します。最後の行では、addListener() を使用してリスナーを Key オブジェクトに登録し、キーが押されたか、離されたときにイベント通知を受け取ることができます。

```
var myListener:Object = new Object();  
myListener.onKeyDown = function () {  
    trace ("You pressed a key.");  
}  
myListener.onKeyUp = function () {  
    trace ("You released a key.");  
}  
Key.addListener(myListener);
```

関連項目

getCode (Key.getCode メソッド), isDown (Key.isDown メソッド), onKeyDown
(Key.onKeyDown イベントリスナー), onKeyUp (Key.onKeyUp イベントリスナー),
removeListener (Key.removeListener メソッド)

BACKSPACE (Key.BACKSPACE プロパティ)

public static BACKSPACE : Number

Backspace キーのキーコード値(8)。

例

次の例では、新しいリスナーオブジェクトを作成し、onKeyDown の関数を定義します。最後の行では、addListener() を使用してリスナーを Key オブジェクトに登録し、キーが押されたときにイベント通知を受け取ることができるようになります。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.BACKSPACE)) {
        trace("you pressed the Backspace key.");
    } else {
        trace("you DIDN'T press the Backspace key.");
    }
};
Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

CAPSLOCK (Key.CAPSLOCK プロパティ)

public static CAPSLOCK : Number

CapsLock キーのキーコード値(20)。

CONTROL (Key.CONTROL プロパティ)

public static CONTROL : Number

Control キーのキーコード値(17)。

DELETEKEY (Key.DELETEKEY プロパティ)

public static DELETEKEY : Number

Delete キーのキーコード値(46)。

例

次の例では、描画 API とリスナーオブジェクトを使用して、マウスカーソルで線を描画できるようになります。描画した線を削除するには、BackSpace キーまたは Delete キーを押します。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
```

```

mouseListener.onMouseDown = function() {
    this.drawing = true;
    canvas_mc.moveTo(_xmouse, _ymouse);
    canvas_mc.lineStyle(3, 0x99CC00, 100);
};

mouseListener.onMouseUp = function() {
    this.drawing = false;
};

mouseListener.onMouseMove = function() {
    if (this.drawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};

Mouse.addListener(mouseListener);
// 
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.DELETEKEY) || Key.isDown(Key.BACKSPACE)) {
        canvas_mc.clear();
    }
};

Key.addListener(keyListener);

この例を使用する場合、テスト環境では [ 制御 ]-[ キーボードショートカットを無効 ] を選択してください。

```

DOWN (Key.DOWN プロパティ)

public static DOWN : Number

下矢印キーのキーコード値 (40)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離 (10) だけ移動します。スペースバーが押されたときはサウンドを再生します。この例で使用するライブラリ内のサウンドに対して、リンクエージ識別子 horn_id を割り当ててください。

```

var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
    }
};

```

```
        break;
    case Key.UP :
        car_mc._y -= DISTANCE;
        break;
    case Key.RIGHT :
        car_mc._x += DISTANCE;
        break;
    case Key.DOWN :
        car_mc._y += DISTANCE;
        break;
    }
};

Key.addListener(keyListener_obj);
```

END (Key.END プロパティ)

public static END : Number

End キーのキーコード値(35)。

ENTER (Key.ENTER プロパティ)

public static ENTER : Number

Enter キーのキーコード値(13)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離(10)だけ移動します。Enter キーが押されると、インスタンス car_mc が停止し、onEnterFrame イベントが削除されます。

```
var DISTANCE:Number = 5;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.LEFT :
            car_mc.onEnterFrame = function() {
                this._x -= DISTANCE;
            };
            break;
        case Key.UP :
            car_mc.onEnterFrame = function() {
                this._y -= DISTANCE;
            };
            break;
        case Key.RIGHT :
            car_mc.onEnterFrame = function() {
                this._x += DISTANCE;
```

```
};

break;
case Key.DOWN :
car_mc.onEnterFrame = function() {
    this._y += DISTANCE;
};
break;
case Key.ENTER :
delete car_mc.onEnterFrame;
break;
}
};

Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

ESCAPE (Key.ESCAPE プロパティ)

public static ESCAPE : Number

Escape キーのキーコード値(27)。

例

次の例では、タイマーを設定します。Esc キーが押されると、キーを押してから離すまでにかかった時間を含む情報を [出力] パネルに表示します。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.ESCAPE)) {
        // get the current timer, convert the value to seconds and round it to two
        decimal places.
        var timer:Number = Math.round(getTimer()/10)/100;
        trace("you pressed the Esc key: "+getTimer()+" ms ("+timer+" s)");
    }
};

Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

getAscii (Key.getAscii メソッド)

```
public static getAscii() : Number
```

最後に押されたか離されたキーの ASCII コードを返します。ASCII の戻り値は英語のキーボード値です。たとえば、Shift + 2 が押されると、Key.getAscii() は日本語のキーボードでも英語のキーボードと同じように @ を返します。

戻り値

Number - 最後に押されたキーの ASCII 値。このメソッドは、押されたり、離されたりしたキーがなかった場合、またはセキュリティ上の理由で ASCII 値にアクセスできない場合は、0 を返します。

例

次の例では、キーが押されるたびに getAscii() メソッドを呼び出します。リスナーオブジェクト keyListener を作成し、onKeyDown イベントの発生時に Key.getAscii() を呼び出す関数を定義します。次に、keyListener オブジェクトを Key オブジェクトに登録します。keyListener は、SWF ファイルの再生中にキーが押されるたびに onKeyDown メッセージをブロードキャストします。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("The ASCII code for the last key typed is: "+Key.getAscii());
};
Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

次の例では、Key.getAscii() の呼び出しを追加して、getAscii() と getCode() の相違点を示します。最も大きな違いは、Key.getAscii() では大文字と小文字が区別されるのに対し、Key.getCode() では大文字と小文字が区別されることです。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
    trace("");
};
Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

getCode (Key.getCode メソッド)

```
public static getCode(): Number
```

最後に押されたキーのキーコード値を返します。

メモ: このメソッドの Flash Lite 版は、プラットフォームによって渡されるキーコードに応じて、ストリングまたは数字を返します。有効なキーコードは、このクラスが受け付ける標準キーコードと ExtendedKey クラスのプロパティとしてリストアップされている特殊キーのコードだけです。

戻り値

Number - 最後に押されたキーのキーコード。このメソッドは、押されたり、離されたりしたキーがなかった場合、またはセキュリティ上の理由でキーコードにアクセスできない場合は、0 を返します。

例

次の例では、キーが押されるたびに getCode() メソッドを呼び出します。リスナーオブジェクト keyListener を作成し、onKeyDown イベントの発生時に Key.getCode() を呼び出す関数を定義します。次に、keyListener オブジェクトを Key オブジェクトに登録します。keyListener は、SWF ファイルの再生中にキーが押されるたびに onKeyDown メッセージをブロードキャストします。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    // Compare return value of getCode() to constant
    if (Key.getCode() == Key.ENTER) {
        trace ("Virtual key code: "+Key.getCode()+" (ENTER key)");
    }
    else {
        trace("Virtual key code: "+Key.getCode());
    }
};
Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

次の例では、Key.getAscii() の呼び出しを追加して、2つのメソッドの相違点を示します。最も大きな違いは、Key.getAscii() では大文字と小文字が区別されるのに対し、Key.getCode() では大文字と小文字が区別されることです。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
    trace("");
};
Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

関連項目

[getAscii \(Key.getAscii メソッド\)](#)

HOME (Key.HOME プロパティ)

public static HOME : Number

Home キーのキーコード値(36)。

例

次の例では、ドラッグ可能なムービークリップ car_mc を x と y 座標 (0,0) に割り当てます。Home キーが押されると、car_mc は (0,0) に戻ります。この例を使用する場合は、リンクエージ識別子が car_id のムービークリップを作成し、タイムラインのフレーム1に次の ActionScript を追加してください。

```
this.attachMovie("car_id", "car_mc", this.getNextHighestDepth(), {_x:0, _y:0});
car_mc.onPress = function() {
    this.startDrag();
};
car_mc.onRelease = function() {
    this.stopDrag();
};
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.HOME)) {
        car_mc._x = 0;
        car_mc._y = 0;
    }
};
Key.addListener(keyListener);
```

INSERT (Key.INSERT プロパティ)

public static INSERT : Number

Insert キーのキーコード値(45)。

例

次の例では、新しいリスナオブジェクトを作成し、onKeyDown の関数を定義します。最後の行では、addListener() を使用してリスナーを Key オブジェクトに登録し、キーが押されたときにイベント通知を受け取り、[出力] パネルに情報を表示できるようにします。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.INSERT)) {
        trace("You pressed the Insert key.");
    }
}
```

```
};  
Key.addListener(keyListener);
```

isDown (Key.isDown メソッド)

public static isDown(code:Number) : Boolean

code に指定されたキーが押された場合は true、それ以外の場合は false を返します。

パラメータ

code:Number - 特定のキーに割り当てられたキーコード値、または特定のキーに関連付けられた Key クラスプロパティ。

戻り値

Boolean - code に指定されたキーが押された場合は true、それ以外の場合は false を返します。

例

次のスクリプトは、ユーザーがムービークリップ (car_mc) の位置を制御できるようにします。

```
car_mc.onEnterFrame = function() {  
    if (Key.isDown(Key.RIGHT)) {  
        this._x += 10;  
    } else if (Key.isDown(Key.LEFT)) {  
        this._x -= 10;  
    }  
};
```

LEFT (Key.LEFT プロパティ)

public static LEFT : Number

左矢印キーのキーコード値 (37)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離 (10) だけ移動します。スペースバーが押されたときはサウンドを再生します。この例で使用するライブラリ内のサウンドに対して、リンクエージ識別子 horn_id を割り当ててください。

```
var DISTANCE:Number = 10;  
var horn_sound:Sound = new Sound();  
horn_sound.attachSound("horn_id");  
var keyListener_obj:Object = new Object();  
keyListener_obj.onKeyDown = function() {  
    switch (Key.getCode()) {  
        case Key.SPACE :  
            horn_sound.start();
```

```
        break;
    case Key.LEFT :
        car_mc._x -= DISTANCE;
        break;
    case Key.UP :
        car_mc._y -= DISTANCE;
        break;
    case Key.RIGHT :
        car_mc._x += DISTANCE;
        break;
    case Key.DOWN :
        car_mc._y += DISTANCE;
        break;
    }
};

Key.addListener(keyListener_obj);
```

_listeners (Key._listeners プロパティ)

public static _listeners : [Array](#) (読み取り専用)

Key オブジェクトに登録したすべてのリスナーオブジェクトへの参照のリスト。このプロパティは内部処理のためのものですが、Key オブジェクトに現時点で登録されているリスナー数を確認する場合に便利なことがあります。addListener() メソッドおよび removelistener() メソッドを呼び出すことで、この配列にオブジェクトを追加したり、この配列からオブジェクトを削除したりします。

例

次の例では、length プロパティを使用して、Key オブジェクトに現時点で登録されているリスナーオブジェクト数を確認します。

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
Key.addListener(myListener);

trace(Key._listeners.length); // Output: 1
```

onKeyDown (Key.onKeyDown イベントリスナー)

```
onKeyDown = function() {}
```

キーを押すと通知されます。onKeyDown を使用するには、リスナーオブジェクトを作成する必要があります。その後、onKeyDown 用の関数を定義し、addListener() を使用してリスナーを Key オブジェクトに登録します。次に例を示します。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tASCII: "+Key.getAscii()+"\tKey:
    "+chr(Key.getAscii()));
};

keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tASCII: "+Key.getAscii()+"\tKey:
    "+chr(Key.getAscii()));
};

Key.addListener(keyListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができますので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

関連項目

[addListener \(Key.addListener メソッド\)](#)

onKeyUp (Key.onKeyUp イベントリスナー)

```
onKeyUp = function() {}
```

キーを離すと通知されます。onKeyUp を使用するには、リスナーオブジェクトを作成する必要があります。その後、onKeyUp 用の関数を定義し、addListener() を使用してリスナーを Key オブジェクトに登録します。次に例を示します。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tASCII: "+Key.getAscii()+"\tKey:
    "+chr(Key.getAscii()));
};

keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tASCII: "+Key.getAscii()+"\tKey:
    "+chr(Key.getAscii()));
};

Key.addListener(keyListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができますので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

関連項目

[addListener \(Key.addListener メソッド\)](#)

PGDN (Key.PGDN プロパティ)

public static PGDN : Number

PageDown キーのキーコード値(34)。

例

次の例では、PageDown キーまたは PageUp キーが押されたときにムービークリップ car_mc を回転させます。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.PGDN)) {
        car_mc._rotation += 5;
    } else if (Key.isDown(Key.PGUP)) {
        car_mc._rotation -= 5;
    }
};
Key.addListener(keyListener);
```

PGUP (Key.PGUP プロパティ)

public static PGUP : Number

PageUp キーのキーコード値(33)。

例

次の例では、PageDown キーまたは PageUp キーが押されたときにムービークリップ car_mc を回転させます。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.PGDN)) {
        car_mc._rotation += 5;
    } else if (Key.isDown(Key.PGUP)) {
        car_mc._rotation -= 5;
    }
};
Key.addListener(keyListener);
```

removeListener (Key.removeListener メソッド)

public static removeListener(listener:[Object](#)) : Boolean

Key.addListener() を使用して以前に登録したオブジェクトを削除します。

パラメータ

listener:[Object](#) - オブジェクト。

戻り値

[Boolean](#) - listener が正常に削除された場合は true を返します。listener が Key オブジェクトのリスナーリストになかった場合など、listener が正常に削除されなかった場合は false を返します。

例

次の例では、左矢印キーと右矢印キーを使用して、ムービークリップ car_mc を移動します。Esc キーを押すとリスナーが削除されるので、car_mc は移動しなくなります。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.LEFT :
            car_mc._x -= 10;
            break;
        case Key.RIGHT :
            car_mc._x += 10;
            break;
        case Key.ESCAPE :
            Key.removeListener(keyListener);
    }
};
Key.addListener(keyListener);
```

RIGHT (Key.RIGHT プロパティ)

public static RIGHT : [Number](#)

右矢印キーのキーコード値(39)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離(10)だけ移動します。スペースバーが押されたときはサウンドを再生します。この例で使用するライブラリ内のサウンドに対して、リンクエージェント horn_id を割り当ててください。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
```

```
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);
```

SHIFT (Key.SHIFT プロパティ)

public static SHIFT : Number

Shift キーのキーコード値(16)。

例

次の例では、Shift キーが押されたときに car_mc を拡大します。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.SHIFT)) {
        car_mc._xscale = 2;
        car_mc._yscale = 2;
    } else if (Key.isDown(Key.CONTROL)) {
        car_mc._xscale /= 2;
        car_mc._yscale /= 2;
    }
};
Key.addListener(keyListener);
```

SPACE (Key.SPACE プロパティ)

```
public static SPACE : Number
```

スペースバーのキーコード値(32)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離(10)だけ移動します。スペースバーが押されたときはサウンドを再生します。この例で使用するライブラリ内のサウンドに対して、リンクエージ識別子 horn_id を割り当ててください。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);
```

TAB (Key.TAB プロパティ)

```
public static TAB : Number
```

Tab キーのキーコード値(9)。

例

次の例では、テキストフィールドを作成し、Tab キーが押されたときに日付をテキストフィールドに表示します。

```
this.createTextField("date_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
date_txt.autoSize = true;
var keyListener:Object = new Object();
```

```
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.TAB)) {
        var today_date:Date = new Date();
        date_txt.text = today_date.toString();
    }
};

Key.addListener(keyListener);
```

この例を使用する場合、テスト環境では [制御]-[キーボードショートカットを無効] を選択してください。

UP (Key.UP プロパティ)

public static UP : Number

上矢印キーのキーコード値(38)。

例

次の例では、矢印キーが押されたときにムービークリップ car_mc を一定の距離(10)だけ移動します。スペースバーが押されたときはサウンドを再生します。この例で使用するライブラリ内のサウンドに対して、リンクエージ識別子 horn_id を割り当ててください。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);
```

LoadVars

```
Object
 |
 +-LoadVars
```

```
public dynamic class LoadVars
extends Object
```

LoadVars クラスを使用すると、データのロードが成功したかどうかを確認することや、ダウンロードの進行状況を監視できます。loadVariables() 関数の代わりに LoadVars クラスを使用して、Flash アプリケーションとサーバーの間で変数を転送できます。

具体的には、LoadVars クラスはオブジェクト内のすべての変数を指定の URL に送ったり、指定された URL にあるすべての変数をオブジェクトにロードしたりできます。また、すべての変数ではなく特定の変数を送信することもできるので、アプリケーションの効率が向上します。LoadVars.onLoad ハンドラを使用すると、(データがロードされる前ではなく) データがロードされるときにアプリケーションが実行されるようにすることができます。

LoadVars クラスの動作は XML クラスとよく似ています。このクラスは、load() メソッド、send() メソッド、および sendAndLoad() メソッドを使用してサーバーと通信します。LoadVars クラスと XML クラスの大きな違いは、LoadVars クラスが ActionScript の名前と値のペアを転送するのに対して、XML クラスは XML オブジェクトに格納されている XML DOM ツリーを転送するということです。LoadVars クラスは、XML クラスと同じセキュリティ制限に従います。

関連項目

[loadVariables 関数](#), [onLoad \(LoadVars.onLoad ハンドラ\)](#), [XML](#)

プロパティ一覧

オプション	プロパティ	説明
	contentType:String	LoadVars.send() または LoadVars.sendAndLoad() を呼び出したときにサーバーに送られる MIME タイプ。
	loaded:Boolean	load 处理または sendAndLoad 处理が完了したかどうかを示す布尔値です。デフォルトは undefined です。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
<code>onData = function (src:String) {}</code>	サーバーからのデータのダウンロードが完了したとき、またはサーバーからデータをダウンロード中にエラーが発生したときに呼び出されます。
<code>onLoad = function(success: Boolean) {}</code>	LoadVars.load() 処理または LoadVars.sendAndLoad() 処理が完了したときに呼び出されます。

コンストラクター一覧

シグネチャ	説明
<code>LoadVars()</code>	LoadVars オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>addRequestHeader (header:Object, headerValue:String) : Void</code>	POST アクションによって送信される HTTP リクエストヘッダ(Content-Type や SOAPAction など)を追加または変更します。
	<code>decode(queryString: String) : Void</code>	変数ストリングを、指定された LoadVars オブジェクトのプロパティに変換します。
	<code>getBytesLoaded() : Number</code>	LoadVars.load() または LoadVars.sendAndLoad() によってダウンロードされたバイト数を返します。
	<code>getBytesTotal() : Number</code>	LoadVars.load() または LoadVars.sendAndLoad() によってダウンロードされた総バイト数を返します。
	<code>load(url:String) : Boolean</code>	指定された URL から変数をダウンロードし、変数データを解析し、結果として得られた変数を my_lv に代入します。
	<code>send(url:String, target:String, [method:String]) : Boolean</code>	<i>my_lv</i> オブジェクト内の変数を、指定された URL に送信します。
	<code>sendAndLoad (url:String, target:Object, [method:String]) : Boolean</code>	<i>my_lv</i> オブジェクト内の変数を、指定された URL に送信(Post)します。
	<code>toString() : String</code>	<i>my_lv</i> 内の列挙可能な変数をすべて含むストリングを、MIME コンテンツエンコード <i>application/x-www-form-urlencoded</i> で返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addRequestHeader (LoadVars.addRequestHeader メソッド)

public addRequestHeader(header:Object, headerValue:String) : Void

POST アクションによって送信される HTTP リクエストヘッダ (Content-Type や SOAPAction など) を追加または変更します。シンタックス 1 では、header および headerValue の 2 つのストリングをメソッドに渡します。シンタックス 2 では、ヘッダー名とヘッダー値を交互に含むストリングの配列を渡します。

同じヘッダー名に対して複数の呼び出しを実行すると、呼び出しのたびに前の呼び出しで設定された値が上書きされます。

このメソッドを使用して、標準の HTTP ヘッダー (Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning および WWW-Authenticate) を追加または変更することはできません。

パラメータ

header:Object - HTTP リクエストヘッダー名を表すストリング、またはストリングの配列。

headerValue:String - header に関連付けられた値を表すストリング。

例

次の例では、値が Foo である SOAPAction というカスタム HTTP ヘッダーを my_lv オブジェクトに追加します。

```
my_lv.addRequestHeader("SOAPAction", "'Foo');
```

次の例では、headers という配列を作成します。この配列には、HTTP ヘッダーとその値が交互に格納されます。この配列を addRequestHeader() に引数として渡します。

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];  
my_lv.addRequestHeader(headers);
```

次の例では、FLASH-UUID というリクエストヘッダーを追加する新しい LoadVars オブジェクトを作成します。ヘッダーには変数が格納され、サーバー側でチェックできます。

```
var my_lv:LoadVars = new LoadVars();
my_lv.addRequestHeader("FLASH-UUID", "41472");
my_lv.name = "Mort";
my_lv.age = 26;
my_lv.send("http://flash-mx.com/mm/cgivars.cfm", "_blank", "POST");
```

関連項目

[addRequestHeader \(XML.addRequestHeader メソッド\)](#)

contentType (LoadVars.contentType プロパティ)

public contentType : String

LoadVars.send() または LoadVars.sendAndLoad() を呼び出したときにサーバーに送られる MIME タイプ。デフォルトは application/x-www-form-urlencoded です。

例

次の例では、LoadVars オブジェクトを作成し、サーバーに送信されるデータのデフォルトのコンテンツタイプを表示します。

```
var my_lv:LoadVars = new LoadVars();
trace(my_lv.contentType); // output: application/x-www-form-urlencoded
```

関連項目

[send \(LoadVars.send メソッド\), sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#)

decode (LoadVars.decode メソッド)

public decode(queryString:String) : Void

変数ストリングを、指定された LoadVars オブジェクトのプロパティに変換します。

このメソッドは、LoadVars.onData イベントハンドラによって内部的に使用されます。ほとんどの場合、ユーザーが直接このメソッドを呼び出す必要はありません。LoadVars.onData イベントハンドラを上書きした場合は、LoadVars.decode() を明示的に呼び出して変数のストリングを解析できます。

パラメータ

queryString:String - 名前と値のペアを含む、URL エンコードされたクエリーストリング。

例

次の例では、2つの変数をトレースしています。

```
// Create a new LoadVars object
var my_lv:LoadVars = new LoadVars();
//Convert the variable string to properties
my_lv.decode("name=Mort&score=250000");
trace(my_lv.toString());
// Iterate over properties in my_lv
for (var prop in my_lv) {
    trace(prop+" -> "+my_lv[prop]);
}
```

関連項目

[onData \(LoadVars.onData ハンドラ\)](#), [parseXML \(XML.parseXML メソッド\)](#)

getBytesLoaded (LoadVars.getBytesLoaded メソッド)

public getBytesLoaded():Number

LoadVars.load() または LoadVars.sendAndLoad() によってダウンロードされたバイト数を返します。ロード処理が実行中ではない場合、またはまだ始まっていない場合は、undefined を返します。

戻り値

Number - 整数。

例

次の例では、ProgressBar インスタンスおよび LoadVars オブジェクトを使用して、テキストファイルをダウンロードします。ファイルをテストすると、ファイルのロードの成否と SWF ファイルにロードされたデータ量が [出力] パネルに表示されます。LoadVars.load() コマンドの URL パラメータは、HTTP を使用して有効なテキストファイルを指定するように値を置き換える必要があります。この例を使用してハードディスク上にあるローカルファイルをロードしようとしても、ムービープレビュー モードの Flash Player ではローカルファイル全体がロードされるため、正常に機能しません。このコードの動作を確認するには、ProgressBar インスタンス loadvars_pb をステージに追加します。次に、タイムラインのフレーム 1 に次の ActionScript を追加します。

```
var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {
    var lvBytesLoaded:Number = my_lv.getBytesLoaded();
    var lvBytesTotal:Number = my_lv.getBytesTotal();
    if (lvBytesTotal != undefined) {
```

```

        trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
        loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
    }
};

my_lv.onLoad = function(success:Boolean) {
    loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
    delete timer_mc.onEnterFrame;
    if (success) {
        trace("LoadVars loaded successfully.");
    } else {
        trace("An error occurred while loading variables.");
    }
};
my_lv.load("[place a valid URL pointing to a text file here]");

```

関連項目

[load \(LoadVars.load メソッド\)](#), [sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#)

getBytesTotal (LoadVars.getBytesTotal メソッド)

public getBytesTotal() : Number

LoadVars.load() または LoadVars.sendAndLoad() によってダウンロードされた総バイト数を返します。ロード処理が実行中ではない場合、またはまだ始まっていない場合には、undefined を返します。総バイト数を判別できない場合(ダウンロードは開始したが、サーバーがHTTPのコンテンツ長を送信しなかった場合など)にも undefined を返します。

戻り値

Number - 整数。

例

次の例では、ProgressBar インスタンスおよび LoadVars オブジェクトを使用して、テキストファイルをダウンロードします。ファイルをテストすると、ファイルのロードの成否と SWF ファイルにロードされたデータ量が [出力] パネルに表示されます。LoadVars.load() コマンドの URL パラメータは、HTTP を使用して有効なテキストファイルを指定するように値を置き換える必要があります。この例を使用してハードディスク上にあるローカルファイルをロードしようとしても、ムービープレビュー モードの Flash Player ではローカルファイル全体がロードされるため、正常に機能しません。このコードの動作を確認するには、ProgressBar インスタンス loadvars_pb をステージに追加します。次に、タイムラインのフレーム 1 に次の ActionScript を追加します。

```

var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {

```

```

var lvBytesLoaded:Number = my_lv.getBytesLoaded();
var lvBytesTotal:Number = my_lv.getBytesTotal();
if (lvBytesTotal != undefined) {
    trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
    loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
}
};

my_lv.onLoad = function(success:Boolean) {
    loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
    delete timer_mc.onEnterFrame;
    if (success) {
        trace("LoadVars loaded successfully.");
    } else {
        trace("An error occurred while loading variables.");
    }
};
my_lv.load("[place a valid URL pointing to a text file here]");

```

関連項目

[load \(LoadVars.load メソッド\)](#), [sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#)

load (LoadVars.load メソッド)

`public load(url:String) : Boolean`

指定された URL から変数をダウンロードし、変数データを解析し、結果として得られた変数を `my_lv` に代入します。`my_lv` のプロパティのうち、ダウンロードされた変数と名前が同じものは上書きされます。`my_lv` のプロパティのうち、ダウンロードされた変数と名前が異なるものは削除されません。これは非同期アクションです。

ダウンロードされたデータは、MIME コンテンツタイプ `application/x-www-form-urlencoded` である必要があります。

これは `loadVariables()` で使用されるのと同じ形式です。

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、`url` は、呼び出し元の SWF ファイルと同じスープードメインに属している必要があります。スープードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、`www.someDomain.com` に存在する SWF ファイルは、`store.someDomain.com` に存在するソースからデータをロードできます。これは、どちらのファイルも同じスープードメイン `someDomain.com` に属しているためです。

Flash Player 7 以降で SWF ファイルを実行している場合、`url` は正確に同じドメインに置かれている必要があります。たとえば `www.someDomain.com` に置かれている SWF ファイルは、

`www.someDomain.com` に置かれているソースからのみデータをロードできます。異なるドメインからデータをロードする場合は、SWF ファイルをホストするサーバーに "クロスドメインポリシー ファイル" を置いておく必要があります。

また、Flash Player 7 用にパブリッシュされたファイルでは、`LoadVars.load()` でロードされる外部変数の大文字と小文字の区別に対応しています。

このメソッドは `XML.load()` と似ています。

パラメータ

`url:String` - スtring。変数のダウンロード元の URL。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、`url` は SWF ファイルと同じドメインに属している必要があります。詳細については、「説明」を参照してください。

戻り値

`Boolean` - パラメータが渡されなかった場合(パラメータが `null` の場合)は `false`、それ以外の場合は `true` を返します。データのロードに成功したかどうかを確認するには、`onLoad()` イベントハンドラを使用します。

例

次のコードでは、サーバー側の PHP スクリプトから Flash アプリケーションにデータが返されたことを通知する `onLoad` ハンドラ関数を定義した後、データを `passvars.php` にロードします。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    if (success) {
        trace(this.toString());
    } else {
        trace("Error loading/parsing LoadVars.");
    }
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "guestbook.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。
```

関連項目

[load \(XML.load メソッド\)](#), [loaded \(LoadVars.loaded プロパティ\)](#),
[onLoad \(LoadVars.onLoad ハンドラ\)](#)

loaded (LoadVars.loaded プロパティ)

```
public loaded : Boolean
```

load 処理または sendAndLoad 処理が完了したかどうかを示すブール値です。デフォルトは undefined です。LoadVars.load() または LoadVars.sendAndLoad() の処理が始まると、loaded プロパティは false に設定されます。処理が完了していない場合、またはエラーが発生して失敗した場合は、loaded プロパティは false に設定されたままです。

このプロパティは XML.loaded プロパティに似ています。

例

次の例では、テキストファイルをロードし、処理が完了したときに情報を [出力] パネルに表示します。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    trace("LoadVars loaded successfully: "+this.loaded);
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

関連項目

[load \(LoadVars.load メソッド\)](#), [sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#),
[load \(XML.load メソッド\)](#)

LoadVars コンストラクタ

```
public LoadVars()
```

LoadVars オブジェクトを作成します。その LoadVars オブジェクトのメソッドを使用してデータを送信およびロードできます。

例

次の例では、my_lv という LoadVars オブジェクトを作成します。

```
var my_lv:LoadVars = new LoadVars();
```

onData (LoadVars.onData ハンドラ)

```
onData = function(src:String) {}
```

サーバーからのデータのダウンロードが完了したとき、またはサーバーからデータをダウンロード中にエラーが発生したときに呼び出されます。このハンドラはデータが解析される前に呼び出されるので、Flash Player に組み込まれている解析ルーチンではなく、独自の解析ルーチンを呼び出す場合に利用できます。LoadVars.onData に割り当てられた関数の src パラメータの値は undefined であるか、またはサーバーからダウンロードされた URL エンコード形式の名前と値のペアを含むストリングです。src パラメータが undefined である場合は、サーバーからのデータのダウンロード時にエラーが発生したことを示しています。

デフォルトの実装では、LoadVars.onData は LoadVars.onLoad を呼び出します。

LoadVars.onData にカスタム関数を割り当ててデフォルトの実装を上書きすることはできますが、独自に実装した LoadVars.onData の中で呼び出さない限り、LoadVars.onLoad は呼び出されません。

パラメータ

src:String - スtringing または undefined。LoadVars.load() メソッドまたは LoadVars.sendAndLoad() メソッドの呼び出しから返される生(未解析)のデータ。

例

次の例では、テキストファイルをロードし、ロード処理の完了時に内容を TextArea インスタンス content_ta に表示します。エラーが発生した場合は、その情報を [出力] パネルに表示します。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src == undefined) {
        trace("Error loading content.");
        return;
    }
    content_ta.text = src;
};
my_lv.load("content.txt", my_lv, "GET");
```

関連項目

[onLoad \(LoadVars.onLoad ハンドラ\)](#), [onLoad \(LoadVars.onLoad ハンドラ\)](#),
[load \(LoadVars.load メソッド\)](#), [sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#)

onLoad (LoadVars.onLoad ハンドラ)

```
onLoad = function(success:Boolean) {}
```

LoadVars.load() 处理または LoadVars.sendAndLoad() 处理が完了したときに呼び出されます。処理が成功した場合は、その処理によってダウンロードされた変数が my_lv に格納されています。これらの変数は、このハンドラが呼び出されると使用できるようになります。

このハンドラはデフォルトでは定義されていません。

このイベントハンドラは XML.onLoad と似ています。

パラメータ

success:Boolean - ブール値。ロード処理が正常に完了した場合は true、正常に完了しなかった場合は false に設定されます。

例

次の例では、TextInput インスタンス name_ti、TextArea インスタンス result_ta、Button インスタンス submit_button をステージに追加します。ユーザーがこの例の Login ボタンインスタンスをクリックすると、send_lv および result_lv の 2 つの LoadVars オブジェクトが作成されます。send_lv オブジェクトは、name_ti インスタンスから名前をコピーし、そのデータを greeting.cfm に送信します。このスクリプトの実行結果を result_lv オブジェクトにロードし、サーバーの応答を TextArea インスタンス (result_ta) に表示します。タイムラインのフレーム 1 に次の ActionScript を追加します。

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
    var result_lv:LoadVars = new LoadVars();
    result_lv.onLoad = function(success:Boolean) {
        if (success) {
            result_ta.text = result_lv.welcomeMessage;
        } else {
            result_ta.text = "Error connecting to server.";
        }
    };
    var send_lv:LoadVars = new LoadVars();
    send_lv.name = name_ti.text;
    send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
        "POST");
};
submit_button.addEventListener("click", submitListener);

```

より堅牢な例については、www.adobe.com/go/learn_fl_samples_jp にある "login.fla" ファイルを参照してください。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[onLoad \(XML.onLoad ハンドラ\)](#), [loaded \(LoadVars.loaded プロパティ\)](#), [load \(LoadVars.load メソッド\)](#), [sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#)

send (LoadVars.send メソッド)

public send(url:String, target:String, [method:String]) : Boolean

my_lv オブジェクト内の変数を、指定された URL に送信します。*my_lv* 内の列挙可能なすべての変数がデフォルトで *application/x-www-form-urlencoded* 形式のストリングとして 1つに連結され、HTTP の POST メソッドを使用して URL に送信されます。これは、*loadVariables()* で使用されるものと同じ形式です。*HTTP* リクエストヘッダで送信される MIME コンテンツタイプは、*my_lv.contentType* の値、またはデフォルトの *application/x-www-form-urlencoded* です。GET を指定しなければ、POST メソッドが使用されます。

指定した URL のスクリプトまたはアプリケーションが確実に実行されるように、target パラメータを指定する必要があります。target パラメータを省略すると、true が返されますが、スクリプトまたはアプリケーションは実行されません。

次の場合は、send() メソッドが便利です。

- サーバー応答で SWF の内容を置き換える (target パラメータとして "*_self*" を使用)
- サーバー応答を新しいウィンドウに表示する (target パラメータとして "*_blank*" を使用)
- サーバー応答を親フレームまたは最上位のフレームに表示する (target パラメータとして "*_parent*" または "*_top*" を使用)
- サーバー応答を指定したフレームに表示する (フレーム名を target パラメータのストリングとして使用)

send() メソッドの呼び出しが成功すると常に、新しいブラウザウィンドウが開くか、既存のウィンドウまたはフレームの内容が置き換わります。情報をサーバーに送り、新しいウィンドウを開いたり、ウィンドウやフレームの内容を置き換えたりせずに、SWF ファイルの再生を続行する場合は、*LoadVars.sendAndLoad()* を使用してください。

このメソッドは XML.send() と似ています。

Flash のテスト環境では、常に GET メソッドを使用します。POST メソッドを使ってテストする場合は、ブラウザ内で使用してください。

パラメータ

`url:String` - ストリング。変数のアップロード先の URL。

`target:String` - ストリング。応答を表示するブラウザのウィンドウまたはフレーム。特定のウィンドウの名前を入力するか、次の予約されたターゲット名から選択します。

- "`_self`" は、現在のウィンドウ内の現在のフレームを指定します。
- "`_blank`" は、新規ウィンドウを指定します。
- "`_parent`" は、現在のフレームの親を指定します。
- "`_top`" は、現在のウィンドウ内の最上位のフレームを指定します。

`method:String`(オプション) - ストリング。HTTP プロトコルの GET メソッドまたは POST メソッド。デフォルト値は POST です。

戻り値

`Boolean` - ブール値。パラメータが指定されない場合は `false`、それ以外の場合は `true` を返します。

例

次の例では、テキストフィールドから 2 つの値をコピーし、その情報を処理する CFM スクリプトに送信します。スクリプトでは、ユーザーがハイスコアを出したかどうかをチェックして、そのデータをデータベースステーブルに挿入するなどの処理を実行します。

```
var my_lv:LoadVars = new LoadVars();
my_lv.playerName = playerName_txt.text;
my_lv.playerScore = playerScore_txt.text;
my_lv.send("setscore.cfm", "_blank", "POST");
```

関連項目

[sendAndLoad \(LoadVars.sendAndLoad メソッド\)](#), [send \(XML.send メソッド\)](#)

sendAndLoad (LoadVars.sendAndLoad メソッド)

`public sendAndLoad(url:String, target:Object, [method:String]) : Boolean`

`my_lv` オブジェクト内の変数を、指定された URL に送信 (Post) します。サーバーの応答がダウンロードされ、変数データとして解析され、結果の変数が `target` オブジェクトに挿入されます。

変数の送信方法は、`LoadVars.send()` の場合と同じです。変数を `target` にダウンロードする方法は `LoadVars.load()` と同じです。

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、url は、呼び出し元の SWF ファイルと同じスープードメインに属している必要があります。スープードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、www.someDomain.com に存在する SWF ファイルは、store.someDomain.com に存在するソースからデータをロードできます。これは、どちらのファイルも同じスープードメイン someDomain.com に属しているためです。

Flash Player 7 以降で SWF ファイルを実行している場合、url は正確に同じドメインに置かれている必要があります。たとえば www.someDomain.com に置かれている SWF ファイルは、www.someDomain.com に置かれているソースからのみデータをロードできます。異なるドメインからデータをロードする場合は、SWF ファイルをホストするサーバーに "クロスドメインポリシー ファイル" を置いておく必要があります。

このメソッドは XML.sendAndLoad() と似ています。

パラメータ

url:String - ストリング。変数のアップロード先の URL。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、url は SWF ファイルと同じドメインに属している必要があります。

target:Object - ダウンロードされる変数を受け取る LoadVars オブジェクトまたは XML オブジェクト。

method:String (オプション) - ストリング。HTTP プロトコルの GET メソッドまたは POST メソッド。デフォルト値は POST です。

戻り値

Boolean - ブール値。

例

次の例では、TextInput インスタンス name_ti、TextArea インスタンス result_ta、Button インスタンス submit_button をステージに追加します。ユーザーがこの例の Login ボタンインスタンスをクリックすると、send_lv および result_lv の 2 つの LoadVars オブジェクトが作成されます。send_lv オブジェクトは、name_ti インスタンスから名前をコピーし、そのデータを greeting.cfm に送信します。このスクリプトの実行結果を result_lv オブジェクトにロードし、サーバーの応答を TextArea インスタンス (result_ta) に表示します。タイムラインのフレーム 1 に次の ActionScript を追加します。

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
    var result_lv:LoadVars = new LoadVars();
    result_lv.onLoad = function(success:Boolean) {
        if (success) {
            result_ta.text = result_lv.welcomeMessage;
        } else {
            result_ta.text = "Error connecting to server.";
    }
}
```

```
        }
    };
    var send_lv:LoadVars = new LoadVars();
    send_lv.name = name_ti.text;
    send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
    "POST");
};

submit_button.addEventListener("click", submitListener);
```

より堅牢な例については、www.adobe.com/go/learn_fl_samples_jp にある "login.fla" ファイルを参照してください。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[send \(LoadVars.send メソッド\)](#), [load \(LoadVars.load メソッド\)](#), [sendAndLoad \(XML.sendAndLoad メソッド\)](#)

toString (LoadVars.toString メソッド)

public **toString()** : String

my_lv 内の列挙可能な変数をすべて含むストリングを、MIME コンテンツエンコード *application/x-www-form-urlencoded* で返します。

戻り値

[String](#) - ストリング。

例

次の例では、新しい LoadVars() オブジェクトをインスタンス化し、2つのプロパティを作成します。次に、**toString()** を使用して、両方のプロパティを含む URL エンコード形式のストリングを返します。

```
var my_lv:LoadVars = new LoadVars();
my_lv.name = "Gary";
my_lv.age = 26;
trace (my_lv.toString()); //output: age=26&name=Gary
```

Math

```
Object
 |
 +-Math
```

```
public class Math
extends Object
```

Math クラスはトップレベルのクラスで、コンストラクタを実行しなくても、そのメソッドやプロパティを使用できます。

数学定数および関数にアクセスして処理するには、このクラスのメソッドとプロパティを使用します。Math クラスのプロパティとメソッドはすべて静的であり、`Math.method(parameter)` または `Math.constant` というシンタックスを使用して呼び出す必要があります。ActionScript では、定数は倍精度の IEEE-754 浮動小数の最大精度で定義されます。

Math クラスのいくつかのメソッドは、ラジアン単位の角度をパラメータとして使用します。メソッドを呼び出す前に次の式を使用してラジアン値を計算し、計算した値をパラメータとして指定できます。また、式の右辺全体 (degrees には度数で角度を代入) をラジアンパラメータとして指定することもできます。

ラジアン値を計算するには、次の式を使用します。

```
radians = degrees * Math.PI/180
```

次の例では、角度 45 度のサインを計算する式をパラメータとして渡します。

```
Math.sin(45 * Math.PI/180) は Math.sin(.7854) と同じです。
```

プロパティ一覧

オプション	プロパティ	説明
static	<code>E:Number</code>	自然対数の底を表す数学定数で <code>e</code> と表記されるものです。
static	<code>LN10:Number</code>	10 の自然対数を表す数学定数で <code>log10</code> と表記されるものです。近似値は 2.302585092994046 です。
static	<code>LN2:Number</code>	2 の自然対数を表す数学定数で <code>log2</code> と表記されるものです。近似値は 0.6931471805599453 です。
static	<code>LOG10E:Number</code>	10 を底とする定数 <code>e</code> (<code>Math.E</code>) の対数を表す数学定数で <code>log10e</code> と表記されるものです。近似値は 0.4342944819032518 です。
static	<code>LOG2E:Number</code>	2 を底とする定数 <code>e</code> (<code>Math.E</code>) の対数を表す数学定数で <code>log2e</code> と表記されるものです。近似値は 1.442695040888963387 です。
static	<code>PI:Number</code>	円周と円の直径の比を表す数学定数で <code>π</code> と表記されるものです。近似値は 3.141592653589793 です。

オプション	プロパティ	説明
static	SQRT1_2:Number	1/2 の平方根を表す数学定数です。近似値は 0.7071067811865476 です。
static	SQRT2:Number	2 の平方根を表す数学定数です。近似値は 1.4142135623730951 です。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

メソッド一覧

オプション	シグネチャ	説明
static	abs(x:Number) : Number	パラメータ x で指定された数値の絶対値を計算して返します。
static	acos(x:Number) : Number	パラメータ x で指定された数値のアークコサイン(逆余弦)を計算してラジアン単位で返します。
static	asin(x:Number) : Number	パラメータ x で指定された数値のアークサイン(逆正弦)を計算してラジアン単位で返します。
static	atan(tangent: Number) : Number	パラメータ tangent で指定された値がタンジェント(正接)の値になる角度を計算してラジアン単位で返します。
static	atan2(y:Number, x:Number) : Number	円の x 軸 (0,0 は円の中心を示します) から反時計回りに測定した場合に、y/x 座標の角度をラジアン単位で計算して返します。
static	ceil(x:Number) : Number	指定された数値または式を切り上げた値を返します。
static	cos(x:Number) : Number	ラジアン単位で指定された角度のコサイン(余弦)を計算して返します。
static	exp(x:Number) : Number	自然対数の底 (e) を、パラメータ x で指定された指数で累乗した値を返します。
static	floor(x:Number) : Number	パラメータ x で指定された数値または式を切り捨てた値を返します。
static	log(x:Number) : Number	パラメータ x の自然対数を返します。
static	max(x:Number, y:Number) : Number	x と y を評価し、大きいほうの値を返します。

オプション	シグネチャ	説明
static	<code>min(x:Number, y:Number) : Number</code>	x と y を評価し、小さいほうの値を返します。
static	<code>pow(x:Number, y:Number) : Number</code>	x の y 乗を計算して返します。
static	<code>random() : Number</code>	疑似乱数 n を返します ($0 \leq n < 1$)。
static	<code>round(x:Number) : Number</code>	パラメータ x の値を最も近い整数に四捨五入し、値を返します。
static	<code>sin(x:Number) : Number</code>	ラジアン単位で指定された角度のサイン(正弦)を計算して返します。
static	<code>sqrt(x:Number) : Number</code>	指定された数値の平方根を計算して返します。
static	<code>tan(x:Number) : Number</code>	指定された角度のタンジェント(正接)を計算して返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty
(Object.hasOwnProperty メソッド), isPrototypeOf
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド),
registerClass (Object.registerClass メソッド), toString
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

abs (Math.abs メソッド)

public static abs(x:Number) : Number

パラメータ x で指定された数値の絶対値を計算して返します。

パラメータ

`x:Number` - 数値。

戻り値

`Number` - 数値。

例

次の例では、`Math.abs()` を使用して数値の絶対値を返します。 x パラメータ (この例では `num`) の値は変わりません。

```
var num:Number = -12;
```

```
var numAbsolute:Number = Math.abs(num);
trace(num); // output: -12
trace(numAbsolute); // output: 12
```

acos (Math.acos メソッド)

public static acos(x:Number) : Number

パラメータ x で指定された数値のアークコサイン(逆余弦)を計算してラジアン単位で返します。

パラメータ

x:Number - -1.0 ~ 1.0 の数値。

戻り値

Number - 数値。パラメータ x のアークコサインです。

例

次の例では、いくつかの値のアークコサインを表示します。

```
trace(Math.acos(-1)); // output: 3.14159265358979
trace(Math.acos(0)); // output: 1.5707963267949
trace(Math.acos(1)); // output: 0
```

関連項目

[asin \(Math.asin メソッド\)](#), [atan \(Math.atan メソッド\)](#),
[atan2 \(Math.atan2 メソッド\)](#), [cos \(Math.cos メソッド\)](#), [sin \(Math.sin メソッド\)](#),
[tan \(Math.tan メソッド\)](#)

asin (Math.asin メソッド)

public static asin(x:Number) : Number

パラメータ x で指定された数値のアークサイン(逆正弦)を計算してラジアン単位で返します。

パラメータ

x:Number - -1.0 ~ 1.0 の数値。

戻り値

Number - 2 で割った負の π と 2 で割った正の π の間 ($-\pi/2 \sim \pi/2$) の数値。

例

次の例では、いくつかの値のアークサインを表示します。

```
trace(Math.asin(-1)); // output: -1.5707963267949
trace(Math.asin(0)); // output: 0
trace(Math.asin(1)); // output: 1.5707963267949
```

関連項目

[acos \(Math.acos メソッド\)](#), [atan \(Math.atan メソッド\)](#),
[atan2 \(Math.atan2 メソッド\)](#), [cos \(Math.cos メソッド\)](#), [sin \(Math.sin メソッド\)](#),
[tan \(Math.tan メソッド\)](#)

atan (Math.atan メソッド)

public static atan(tangent:Number) : Number

パラメータ tangent で指定された値がタンジェント(正接)の値になる角度を計算してラジアン単位で返します。戻り値は、2で割った負の π と 2で割った正の π の間($-\pi/2 \sim \pi/2$)の値になります。

パラメータ

tangent:Number - 角度のタンジェントを表す数値。

戻り値

Number - 2で割った負の π と 2で割った正の π の間($-\pi/2 \sim \pi/2$)の数値。

例

次の例では、いくつかのタンジェントの角度の値を表示します。

```
trace(Math.atan(-1)); // output: -0.785398163397448
trace(Math.atan(0)); // output: 0
trace(Math.atan(1)); // output: 0.785398163397448
```

関連項目

[acos \(Math.acos メソッド\)](#), [asin \(Math.asin メソッド\)](#),
[atan2 \(Math.atan2 メソッド\)](#), [cos \(Math.cos メソッド\)](#), [sin \(Math.sin メソッド\)](#),
[tan \(Math.tan メソッド\)](#)

atan2 (Math.atan2 メソッド)

public static atan2(y:Number, x:Number) : Number

円の x 軸(0,0 は円の中心を示します)から反時計回りに測定した場合に、y/x 座標の角度をラジアン単位で計算して返します。戻り値は、正の π と負の π の間の値になります。

パラメータ

y:Number - ポイントの y 座標を指定する数値。

x:Number - ポイントの x 座標を指定する数値。

戻り値

Number - 数値。

例

次の例では、座標 (0,10) によって指定されたポイント (たとえば $x=0$ と $y=10$) の角度をラジアンで返します。atan2への最初のパラメータは常に y 座標です。

```
trace(Math.atan2(10, 0)); // output: 1.5707963267949
```

関連項目

[acos \(Math.acos メソッド\)](#), [asin \(Math.asin メソッド\)](#), [atan \(Math.atan メソッド\)](#),
[cos \(Math.cos メソッド\)](#), [sin \(Math.sin メソッド\)](#), [tan \(Math.tan メソッド\)](#)

ceil (Math.ceil メソッド)

public static ceil(x:Number) : Number

指定された数値または式を切り上げた値を返します。数値の切り上げとは、その数値以上の最も近い整数にすることです。

パラメータ

x:Number - 数値または式。

戻り値

Number - パラメータ x の値以上の最も近い整数。

例

次の例では、値 13 を返します。

```
Math.ceil(12.5);
```

関連項目

[floor \(Math.floor メソッド\)](#), [round \(Math.round メソッド\)](#)

cos (Math.cos メソッド)

public static cos(x:Number) : Number

ラジアン単位で指定された角度のコサイン(余弦)を計算して返します。ラジアンを計算するには、[Math クラス](#)の説明を参照してください。

パラメータ

x:Number - 角度をラジアンで表した数値。

戻り値

Number - -1.0 ~ 1.0 の数値。

例

次の例では、いくつかの角度のコサインを表示します。

```
trace (Math.cos(0)); // 0 degree angle. Output: 1
trace (Math.cos(Math.PI/2)); // 90 degree angle. Output: 6.12303176911189e-17
trace (Math.cos(Math.PI)); // 180 degree angle. Output: -1
trace (Math.cos(Math.PI*2)); // 360 degree angle. Output: 1
```

メモ: 90 度のコサインはゼロですが、2 進数を使用する 10 進数計算に伴う誤差により、Flash Player では、ゼロに極めて近い値ですが、ゼロではない数値になります。

関連項目

[acos \(Math.acos メソッド\)](#), [asin \(Math.asin メソッド\)](#), [atan \(Math.atan メソッド\)](#), [atan2 \(Math.atan2 メソッド\)](#), [sin \(Math.sin メソッド\)](#), [tan \(Math.tan メソッド\)](#)

E (Math.E プロパティ)

public static E : Number

自然対数の底を表す数学定数で e と表記されるものです。e の近似値は 2.71828182845905 です。

例

この例では、Math.E を使用して、1 年間に 100 % の利子が付く場合の複利を計算する方法を示します。

```
var principal:Number = 100;
var simpleInterest:Number = 100;
var continuouslyCompoundedInterest:Number = (100 * Math.E) - principal;
```

```
trace ("Beginning principal: $" + principal);
trace ("Simple interest after one year: $" + simpleInterest);
trace ("Continuously compounded interest after one year: $" +
continuouslyCompoundedInterest);

//  
Output:  
Beginning principal: $100  
Simple interest after one year: $100  
Continuously compounded interest after one year: $171.828182845905
```

exp (Math.exp メソッド)

public static exp(x:Number) : Number

自然対数の底 (e) を、パラメータ x で指定された指数で累乗した値を返します。定数 Math.E を使用して、e の値を指定できます。

パラメータ

x:Number - 指数。数値または式です。

戻り値

Number - 数値。

例

次の例では、2つの値の対数を表示します。

```
trace(Math.exp(1)); // output: 2.71828182845905
trace(Math.exp(2)); // output: 7.38905609893065
```

関連項目

[E \(Math.E プロパティ \)](#)

floor (Math.floor メソッド)

public static floor(x:Number) : Number

パラメータ x で指定された数値または式を切り捨てた値を返します。切り捨てとは、指定された数値または式以下の最も近い整数にすることです。

パラメータ

x:Number - 数値または式。

戻り値

`Number` - パラメータ x の値以下の最も近い整数。

例

次の例では、値 12 を返します。

```
Math.floor(12.5);
```

次の例では、値 -7 を返します。

```
Math.floor(-6.5);
```

LN10 (Math.LN10 プロパティ)

```
public static LN10 : Number
```

10 の自然対数を表す数学定数で `log10` と表記されるものです。近似値は 2.302585092994046 です。

例

この例では、`Math.LN10` の値を出力します。

```
trace(Math.LN10);
// output: 2.30258509299405
```

LN2 (Math.LN2 プロパティ)

```
public static LN2 : Number
```

2 の自然対数を表す数学定数で `log2` と表記されるものです。近似値は 0.6931471805599453 です。

log (Math.log メソッド)

```
public static log(x:Number) : Number
```

パラメータ x の自然対数を返します。

パラメータ

`x:Number` - 値が 0 よりも大きい数値または式。

戻り値

`Number` - パラメータ x の自然対数。

例

次の例では、3つの数値の対数を表示します。

```
trace(Math.log(0)); // output: -Infinity  
trace(Math.log(1)); // output: 0  
trace(Math.log(2)); // output: 0.693147180559945  
trace(Math.log(Math.E)); // output: 1
```

LOG10E (Math.LOG10E プロパティ)

```
public static LOG10E : Number
```

10 を底とする定数 e (`Math.E`) の対数を表す数学定数で `log10e` と表記されるものです。近似値は 0.4342944819032518 です。

`Math.log()` メソッドは、数値の自然対数を計算します。`Math.log()` の結果に `Math.LOG10E` を乗算すると、10 を底とする対数を得ることができます。

例

次の例では、10 を底とする対数の計算方法を示します。

```
trace(Math.log(1000) * Math.LOG10E);  
// Output: 3
```

LOG2E (Math.LOG2E プロパティ)

```
public static LOG2E : Number
```

2 を底とする定数 e (`Math.E`) の対数を表す数学定数で `log2e` と表記されるものです。近似値は 1.442695040888963387 です。

`Math.log()` メソッドは、数値の自然対数を計算します。`Math.log()` の結果に `Math.LOG2E` を乗算すると、2 を底とする対数を得ることができます。

例

次の例では、2 を底とする対数の計算方法を示します。

```
trace(Math.log(16) * Math.LOG2E);  
// Output: 4
```

max (Math.max メソッド)

public static max(x:Number, y:Number) : Number

x と y を評価し、大きいほうの値を返します。

パラメータ

x:Number - 数値または式。

y:Number - 数値または式。

戻り値

Number - 数値。

例

次の例では、評価された式のうち大きい方である Thu Dec 30 00:00:00 GMT-0700 2004 を表示します。

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var maxDate:Number = Math.max(date1.getTime(), date2.getTime());
trace(new Date(maxDate).toString());
```

関連項目

[min \(Math.min メソッド \)](#)

min (Math.min メソッド)

public static min(x:Number, y:Number) : Number

x と y を評価し、小さいほうの値を返します。

パラメータ

x:Number - 数値または式。

y:Number - 数値または式。

戻り値

Number - 数値。

例

次の例では、評価された式のうち小さい方である Sat Dec 25 00:00:00 GMT-0700 2004 を表示します。

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var minDate:Number = Math.min(date1.getTime(), date2.getTime());
trace(new Date(minDate).toString());
```

関連項目

[max \(Math.max メソッド\)](#)

PI (Math.PI プロパティ)

public static PI : Number

円周と円の直径の比を表す数学定数で π と表記されるものです。近似値は 3.141592653589793 です。

例

次の例では、数学定数 π と描画 API を使用して円を描きます。

```
drawCircle(this, 100, 100, 50);
{
    function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
        mc.lineStyle(2, 0xFF0000, 100);
        mc.moveTo(x+r, y);
        mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
        Math.sin(Math.PI/4)*r+y);
        mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
        mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
        Math.sin(Math.PI/4)*r+y);
        mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
        mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x,
        -Math.sin(Math.PI/4)*r+y);
        mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
        mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x,
        -Math.sin(Math.PI/4)*r+y);
        mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
    }
}
```

pow(Math.pow メソッド)

```
public static pow(x:Number, y:Number) : Number  
x の y 乗を計算して返します。
```

パラメータ

x:Number - 累乗される数値。底ともいいます。

y:Number - パラメータ x を累乗する指数。

戻り値

Number - 数値。

例

この例では、Math.pow と Math.sqrt を使用して線の長さを計算します。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());  
var mouseListener:Object = new Object();  
mouseListener.onMouseDown = function() {  
    this.origX = _xmouse;  
    this.origY = _ymouse;  
};  
mouseListener.onMouseUp = function() {  
    this.newX = _xmouse;  
    this.newY = _ymouse;  
    var minY = Math.min(this.origY, this.newY);  
    var nextDepth:Number = canvas_mc.getNextHighestDepth();  
    var line_mc:MovieClip =  
        canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);  
    line_mc.moveTo(this.origX, this.origY);  
    line_mc.lineStyle(2, 0x000000, 100);  
    line_mc.lineTo(this.newX, this.newY);  
    var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,  
        2)+Math.pow(line_mc._height, 2));  
    line_mc.createTextField("length"+nextDepth+"_txt",  
        canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);  
    line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) + " pixels";  
};  
Mouse.addListener(mouseListener);
```

random (Math.random メソッド)

public static random() : Number

疑似乱数 n を返します ($0 \leq n < 1$)。返される値は疑似乱数となります。放射性崩壊のような現実のランダムな自然現象で生成されるわけではないからです。

戻り値

Number - 数値。

例

次の例では、4 ~ 11(4 と 11 を含みます) の整数を 100 個出力します。

```
function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.floor(Math.random() * (max - min + 1)) + min;
    return randomNum;
}
for (var i = 0; i < 100; i++) {
    var n:Number = randRange(4, 11)
    trace(n);
}
```

round (Math.round メソッド)

public static round(x:Number) : Number

パラメータ x の値を最も近い整数に四捨五入し、値を返します。パラメータ x が 2 つの最も近い整数から等距離である場合 (.5 で終わる数値など)、値は次に大きな整数に切り上げられます。

パラメータ

x:Number - 数値。

戻り値

Number - 数値。整数です。

例

次の例では、指定された 2 つの整数の間にあるランダムな数値を返します。

```
function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.round(Math.random() * (max-min+1) + (min-.5));
    return randomNum;
}
for (var i = 0; i<25; i++) {
    trace(randRange(4, 11));
}
```

関連項目

[ceil \(Math.ceil メソッド\)](#), [floor \(Math.floor メソッド\)](#)

sin (Math.sin メソッド)

public static sin(x:Number) : Number

ラジアン単位で指定された角度のサイン(正弦)を計算して返します。ラジアンを計算するには、[Math クラス](#)の説明を参照してください。

パラメータ

x:Number - 角度をラジアンで表した数値。

戻り値

Number - 数値。指定された角度のサイン(-1.0 ~ 1.0)です。

例

次の例では、数学定数 π 、角度のサイン、および描画 API を使用して円を描きます。

```
drawCircle(this, 100, 100, 50);
// 
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

関連項目

[acos \(Math.acos メソッド\)](#), [asin \(Math.asin メソッド\)](#), [atan \(Math.atan メソッド\)](#),
[atan2 \(Math.atan2 メソッド\)](#), [cos \(Math.cos メソッド\)](#), [tan \(Math.tan メソッド\)](#)

sqrt (Math.sqrt メソッド)

```
public static sqrt(x:Number) : Number
```

指定された数値の平方根を計算して返します。

パラメータ

x:Number - 0 以上の数値または式。

戻り値

Number - パラメータ x が 0 以上の場合には数値、それ以外は NaN (非数) を返します。

例

この例では、Math.pow と Math.sqrt を使用して線の長さを計算します。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.origX = _xmouse;
    this.origY = _ymouse;
};
mouseListener.onMouseUp = function() {
    this newX = _xmouse;
    this newY = _ymouse;
    var minY = Math.min(this.origY, this.newY);
    var nextDepth:Number = canvas_mc.getNextHighestDepth();
    var line_mc:MovieClip =
        canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);
    line_mc.moveTo(this.origX, this.origY);
    line_mc.lineStyle(2, 0x000000, 100);
    line_mc.lineTo(this.newX, this.newY);
    var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,
    2)+Math.pow(line_mc._height, 2));
    line_mc.createTextField("length"+nextDepth+"_txt",
        canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);
    line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) +" pixels";
};
Mouse.addListener(mouseListener);
```

SQRT1_2 (Math.SQRT1_2 プロパティ)

```
public static SQRT1_2 : Number
```

1/2 の平方根を表す数学定数です。近似値は 0.7071067811865476 です。

例

この例では、Math.SQRT1_2. の値を出力します。

```
trace(Math.SQRT1_2);
// Output: 0.707106781186548
```

SQRT2 (Math.SQRT2 プロパティ)

```
public static SQRT2 : Number
```

2 の平方根を表す数学定数です。近似値は 1.4142135623730951 です。

例

この例では、Math.SQRT2. の値を出力します。

```
trace(Math.SQRT2);
// Output: 1.4142135623731
```

tan (Math.tan メソッド)

```
public static tan(x:Number) : Number
```

指定された角度のタンジェント (正接) を計算して返します。ラジアンを計算するには、Math クラスの概説を参照してください。

パラメータ

x:Number - 角度をラジアンで表した数値。

戻り値

Number - 数値。パラメータ x のタンジェントを返します。

例

次の例では、数学定数 π 、角度のタンジェント、および描画 API を使用して円を描きます。

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
```

```

mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}

```

関連項目

[acos](#) (`Math.acos` メソッド), [asin](#) (`Math.asin` メソッド), [atan](#) (`Math.atan` メソッド),
[atan2](#) (`Math.atan2` メソッド), [cos](#) (`Math.cos` メソッド), [sin](#) (`Math.sin` メソッド)

Mouse

```

Object
|
+-Mouse

```

```

public class Mouse
extends Object

```

Mouse クラスはトップレベルのクラスで、コンストラクタを実行しなくても、そのメソッドやプロパティを使用できます。Mouse クラスのメソッドを使用して、リスナーを追加および削除したり、マウスイベントを処理したりできます。

メモ: このクラスのメンバーは、Flash Lite で、`System.capabilities.hasMouse` が `true` であるか、または `System.capabilities.hasStylus` が `true` である場合にのみサポートされます。

プロパティ一覧

Object クラスから継承されるプロパティ

<code>constructor</code> (<code>Object.constructor</code> プロパティ), <code>__proto__</code> (<code>Object.__proto__</code> プロパティ), <code>prototype</code> (<code>Object.prototype</code> プロパティ), <code>__resolve</code> (<code>Object.__resolve</code> プロパティ)
--

イベント一覧

イベント	説明
<code>onMouseDown = function() {}</code>	マウスボタンが押されると通知されます。
<code>onMouseMove = function() {}</code>	マウスポインタが移動すると通知されます。
<code>onMouseUp = function() {}</code>	マウスボタンが離されると通知されます。

メソッド一覧

オプション	シグネチャ	説明
static	<code>addListener(listener: Object) : Void</code>	onMouseDown、onMouseMove、onMouseUp の各リスナーの通知を受け取るオブジェクトを登録します。
static	<code>removeListener (listener: Object) : Boolean</code>	以前に addListener() を使用して登録したオブジェクトを削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (Mouse.addListener メソッド)

`public static addListener(listener:Object) : Void`

onMouseDown、onMouseMove、onMouseUp の各リスナーの通知を受け取るオブジェクトを登録します。

listener パラメータは、リスナーの1つ以上に対して定義済みメソッドを持つオブジェクトを含む必要があります。

マウスボタンを押すか、移動するか、離すか、またはマウスでスクロールすると、フォーカスの有無に関係なく、このメソッドで登録したすべてのリスナーオブジェクトの onMouseDown メソッド、onMouseMove メソッド、または onMouseUp メソッドが呼び出されます。複数のオブジェクトがマウス通知を受け取ることができます。リスナーが登録済みの場合は何も変化しません。

メモ: このメソッドは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

パラメータ

listener:[Object](#) - オブジェクト。

例

この例は、"ActionScript" サンプルフォルダにある "animation.fla" ファイルから抜粋したものです。

```
// Create a mouse listener object.  
var mouseListener:Object = new Object();  
  
// Every time the mouse cursor moves within the SWF file,  
// update the position of the crosshair movie clip  
// instance on the Stage.  
mouseListener.onMouseMove = function() {  
    crosshair_mc._x = _xmouse;  
    crosshair_mc._y = _ymouse;  
};  
  
// When you press the mouse button, check to see if the cursor is within the  
// boundaries of the Stage. If so, increment the number of shots.  
mouseListener.onMouseDown = function() {  
    if (bg_mc.hitTest(_xmouse, _ymouse, false)) {  
        _global.shots++;  
    }  
};  
Mouse.addListener(mouseListener);
```

スクリプト全体を表示するには、www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "animation.fla" ファイルを参照してください。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[onMouseDown \(Mouse.onMouseDown イベントリスナー\)](#),
[onMouseMove \(Mouse.onMouseMove イベントリスナー\)](#),
[onMouseUp \(Mouse.onMouseUp イベントリスナー\)](#)

onMouseDown (Mouse.onMouseDown イベントリスナー)

```
onMouseDown = function() {}
```

マウスボタンが押されると通知されます。onMouseDown リスナーを使用するには、リスナーオブジェクトを作成します。その後、onMouseDown の関数を定義し、addListener() メソッドを使用してリスナーを Mouse オブジェクトに登録します。次に例を示します。

```
var someListener:Object = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができますので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

メモ: このイベントリスナーは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、描画 API を使用して、ユーザーが実行時にマウスボタンを押すか、マウスを移動してから、マウスボタンを解放するときに矩形を描画します。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    this.orig_x = _xmouse;
    this.orig_y = _ymouse;
    this.target_mc = canvas_mc.createEmptyMovieClip("", 
        canvas_mc.getNextHighestDepth());
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        this.target_mc.clear();
        this.target_mc.lineStyle(1, 0xFF0000, 100);
        this.target_mc.moveTo(this.orig_x, this.orig_y);
        this.target_mc.lineTo(_xmouse, this.orig_y);
        this.target_mc.lineTo(_xmouse, _ymouse);
        this.target_mc.lineTo(this.orig_x, _ymouse);
        this.target_mc.lineTo(this.orig_x, this.orig_y);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

関連項目

[addListener \(Mouse.addListener メソッド\)](#)

onMouseMove (Mouse.onMouseMove イベントリスナー)

```
onMouseMove = function() {}
```

マウスポインタが移動すると通知されます。onMouseMove リスナーを使用するには、リスナーオブジェクトを作成します。その後、onMouseMove の関数を定義し、addListener() メソッドを使用してリスナーを Mouse オブジェクトに登録します。次に例を示します。

```
var someListener:Object = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができますので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

メモ: このイベントリスナーは、System.capabilities.hasMouse が true の場合のみ Flash Lite でサポートされます。

例

次の例では、マウスポインタをツールとして使用し、onMouseMove と Drawing API を使用して線を描画します。ユーザーがマウスポインタを移動すると、線が描かれます。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

次の例では、ムービークリップインスタンス pointer_mc の x 座標と y 座標をポインタの位置の x 座標と y 座標に設定します。このコード例が動作するには、デバイスがスタイルスまたはマウスをサポートしている必要があります。このコード例を使用するには、ムービークリップを作成し、そのリンクエージ識別子を pointer_id に設定します。次に、タイムラインのフレーム 1 に次の ActionScript コードを追加します。

```
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    pointer_mc._x = _xmouse;
    pointer_mc._y = _ymouse;
};
Mouse.addListener(mouseListener);
```

関連項目

[addListener \(Mouse.addListener メソッド\)](#)

onMouseUp (Mouse.onMouseUp イベントリスナー)

```
onMouseUp = function() {}
```

マウスボタンが離されると通知されます。onMouseUp リスナーを使用するには、リスナーオブジェクトを作成します。その後、onMouseUp の関数を定義し、addListener() メソッドを使用してリスナーを Mouse オブジェクトに登録します。次に例を示します。

```
var someListener:Object = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができますので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

メモ: このイベントリスナーは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、マウスポインタをツールとして使用し、onMouseMove と Drawing API を使用して線を描画します。ユーザーがマウスポインタを移動すると線が描かれ、マウスボタンを解放すると線の描画が停止します。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
```

```
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};

mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};

Mouse.addListener(mouseListener);
```

関連項目

[addListener \(Mouse.addListener メソッド\)](#)

removeListener (Mouse.removeListener メソッド)

public static removeListener(listener:[Object](#)) : Boolean

以前に `addListener()` を使用して登録したオブジェクトを削除します。

メモ: このメソッドは、`System.capabilities.hasMouse` が `true` である、または `System.capabilities.hasStylus` が `true` の場合のみ Flash Lite でサポートされます。

パラメータ

`listener:Object` - オブジェクト。

戻り値

`Boolean` - リスナーオブジェクトが正常に削除されると、`true` を返します。リスナーオブジェクトが正常に削除されない場合（たとえば、リスナーオブジェクトが `Mouse` オブジェクトのリスナーリストに登録されていない場合）は、`false` を返します。

例

次の例では、ステージに 3 つのボタンを割り当てて、実行時にユーザーがマウスポインタを使用して SWF ファイルで線を描くことができるようになります。ボタンの 1 つは、SWF ファイルからすべての線を消去します。2 番目のボタンは、マウスリスナーを削除して、ユーザーが線を描けないようにします。3 番目のボタンは、削除したマウスリスナーを追加して、ユーザーが再び線を描けるようにします。タイムラインのフレーム 1 に次の ActionScript を追加します。

```
this.createClassObject(mx.controls.Button, "clear_button",
    this.getNextHighestDepth(), {_x:10, _y:10, label:'clear'});
this.createClassObject(mx.controls.Button, "stopDrawing_button",
    this.getNextHighestDepth(), {_x:120, _y:10, label:'stop drawing'});
this.createClassObject(mx.controls.Button, "startDrawing_button",
    this.getNextHighestDepth(), {_x:230, _y:10, label:'start drawing'});
startDrawing_button.enabled = false;
//
```

```

this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
var clearListener:Object = new Object();
clearListener.click = function() {
    canvas_mc.clear();
};
clear_button.addEventListener("click", clearListener);
//
var stopDrawingListener:Object = new Object();
stopDrawingListener.click = function(evt:Object) {
    Mouse.removeListener(mouseListener);
    evt.target.enabled = false;
    startDrawing_button.enabled = true;
};
stopDrawing_button.addEventListener("click", stopDrawingListener);
var startDrawingListener:Object = new Object();
startDrawingListener.click = function(evt:Object) {
    Mouse.addListener(mouseListener);
    evt.target.enabled = false;
    stopDrawing_button.enabled = true;
};
startDrawing_button.addEventListener("click", startDrawingListener);

```

MovieClip

```

Object
|
+-MovieClip

```

```

public dynamic class MovieClip
extends Object

```

MovieClip クラスのメソッドは、ムービークリップをターゲットとするアクションと同じ機能を提供します。【アクション】パネルの【アクション】ツールボックスに同等のアクションを持たない他のメソッドもあります。

新しいムービークリップを作成する場合は、コンストラクタメソッドを使用しません。新しいムービークリップインスタンスは、次の3つのいずれかのメソッドを使用して作成します。

- `attachMovie()` メソッドを使用すると、ライブラリ内に存在するムービークリップシンボルに基づいて、新しいムービークリップインスタンスを作成できます。
- `createEmptyMovieClip()` メソッドを使用すると、別のムービークリップに基づいて、新しい空のムービークリップインスタンスを子として作成できます。
- `duplicateMovieClip()` メソッドを使用すると、別のムービークリップに基づいて、ムービークリップインスタンスを作成できます。

`MovieClip` クラスのメソッドを呼び出すには、次のシンタックスを使用して、ムービークリップインスタンスを名前で参照します。このシンタックスでは、`my_mc` がムービークリップインスタンスです。

```
my_mc.play();
my_mc.gotoAndPlay(3);
```

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

プロパティ一覧

オプション	プロパティ	説明
	<code>_alpha:Number</code>	ムービークリップのアルファ透明度。
	<code>_currentframe:Number</code> (読み取り専用)	ムービークリップのタイムライン内で再生ヘッドがあるフレーム番号を返します。
	<code>_droptarget:String</code> (読み取り専用)	このムービークリップがドロップされたムービークリップインスタンスの絶対パスを、スラッシュシンタックス表記で返します。
	<code>enabled:Boolean</code>	ムービークリップが有効であるか無効であるかを示すブール値。
	<code>focusEnabled:Boolean</code>	このプロパティが <code>undefined</code> または <code>false</code> の場合、ボタン以外のムービークリップは入力フォーカスを受け取ることができません。
	<code>_focusrect:Boolean</code>	ムービークリップに入力フォーカスがあるときに、その周囲に黄色の矩形を表示するかどうかを指定するブール値。
	<code>_framesloaded:Number</code> (読み取り専用)	ストリーミング SWF ファイルからロードされたフレーム数。
	<code>_height:Number</code>	ピクセル単位で示したムービークリップの高さです。

オプション	プロパティ	説明
	<code>_highquality:Number</code>	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに MovieClip._quality を使用します。 現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。
	<code>hitArea:Object</code>	ムービークリップのヒット領域となる別のムービークリップを指定します。
	<code>_lockroot:Boolean</code>	SWF ファイルがムービークリップにロードされたときに、_root で参照する内容を指定するブール値。
	<code>_name:String</code>	ムービークリップのインスタンス名。
	<code>_parent:MovieClip</code>	現在のボタンを含むムービークリップを指す参照です。
	<code>_quality:String</code>	SWF ファイルに使用するレンダリング品質を設定または取得します。
	<code>_rotation:Number</code>	ムービークリップの元の位置からの回転角度を度数で指定します。
	<code>_soundbuftime:Number</code>	サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。
	<code>tabChildren:Boolean</code>	ムービークリップの子が自動タブ順に含まれているかどうかを判別します。
	<code>tabEnabled:Boolean</code>	ムービークリップが自動タブ順に含まれるかどうかを示します。
	<code>tabIndex:Number</code>	ムービー内のオブジェクトのタブ順をカスタマイズできます。
	<code>_target:String (読み取り専用)</code>	ムービークリップインスタンスのターゲットパスをスラッシュ表記で返します。
	<code>_totalframes:Number (読み取り専用)</code>	MovieClip パラメータで指定されたムービークリップインスタンスのフレームの総数を返します。
	<code>trackAsMenu:Boolean</code>	他のボタンまたはムービークリップがマウスまたはスタイルラスから解放イベントを受け取ることができるかどうかを示すブール値です。
	<code>_url:String (読み取り専用)</code>	ムービークリップのダウンロード元である SWF、JPEG、GIF、または PNG の各ファイルの URL を取得します。
	<code>_visible:Boolean</code>	ムービークリップを表示するかどうかを示すブール値です。
	<code>_width:Number</code>	ピクセル単位で示したムービークリップの幅。

オプション	プロパティ	説明
	<code>_x:Number</code>	親ムービークリップのローカル座標を基準にしてムービークリップの x 座標を設定する整数。
	<code>_xmouse:Number</code> (読み取り専用)	マウス位置の x 座標を返します。
	<code>_xscale:Number</code>	ムービークリップの基準点から適用するムービークリップの水平方向の拡大 / 縮小率 (percentage) を設定します。
	<code>_y:Number</code>	親ムービークリップのローカル座標を基準にしてムービークリップの y 座標を設定します。
	<code>_ymouse:Number</code> (読み取り専用)	マウス位置の y 座標を示します。
	<code>_yscale:Number</code>	ムービークリップの基準点から適用するムービークリップの垂直スケール (percentage) を設定します。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
<code>onData = function() {}</code>	ムービークリップが MovieClip.loadVariables() の呼び出しありまたは MovieClip.loadMovie() の呼び出しからデータを受け取ったときに呼び出されます。
<code>onDragOut = function() {}</code>	マウスボタンが押された状態で、ピントタがオブジェクトの外に移動したときに呼び出されます。
<code>onDragOver = function() {}</code>	ピントタがムービークリップ外にドラッグされた後で、ムービークリップ上に移動したときに呼び出されます。
<code>onEnterFrame = function() {}</code>	SWF ファイルのフレームレートで繰り返し呼び出されます。
<code>onKeyDown = function() {}</code>	ムービークリップにフォーカスがあるときにキーを押すと、呼び出されます。
<code>onKeyUp = function() {}</code>	キーを離すと呼び出されます。

イベント	説明
<code>onKillFocus = function(newFocus: Object) {}</code>	ムービークリップが入力フォーカスを失うと、呼び出されます。
<code>onLoad = function() {}</code>	ムービークリップのインスタンスが作成されてタイムラインに読み込まれると、呼び出されます。
<code>onMouseDown = function() {}</code>	マウスボタンが押されると、呼び出されます。
<code>onMouseMove = function() {}</code>	マウスポインタが移動すると、呼び出されます。
<code>onMouseUp = function() {}</code>	マウスボタンが離されると、呼び出されます。
<code>onPress = function() {}</code>	ポインタがムービークリップ上にあるときにマウスをクリックすると、呼び出されます。
<code>onRelease = function() {}</code>	ムービークリップの上でマウスボタンを離すと、呼び出されます。
<code>onReleaseOutside = function() {}</code>	マウスボタンをムービークリップ領域内で押して、ムービークリップ領域の外側で離したときに呼び出されます。
<code>onRollOut = function() {}</code>	ポインタがムービークリップ領域の外側に移動すると、呼び出されます。
<code>onRollOver = function() {}</code>	ポインタがムービークリップ領域の上に移動すると、呼び出されます。
<code>onSetFocus = function(oldFocus: Object) {}</code>	ムービークリップが入力フォーカスを受け取ると、呼び出されます。
<code>onUnload = function() {}</code>	ムービークリップをタイムラインから削除した後に表示される最初のフレームで呼び出されます。

メソッド一覧

オプション	シグネチャ	説明
	<code>attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip</code>	ライブラリからシンボルを取得し、ムービークリップに割り当てます。
	<code>beginFill(rgb:Number, [alpha:Number]) : Void</code>	新しい描画パスの始点を示します。
	<code>beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object) : Void</code>	新しい描画パスの始点を示します。
	<code>clear() : Void</code>	<code>MovieClip.lineStyle()</code> で指定された線スタイルを含め、ムービークリップの描画メソッドを使用して実行時に作成されたすべてのグラフィックを削除します。
	<code>createEmptyMovieClip(name:String, depth:Number) : MovieClip</code>	既存のムービークリップの子として空のムービークリップを作成します。
	<code>createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number) : TextField</code>	このメソッドを呼び出したムービークリップの子となる空のテキストフィールドを新しく作成します。
	<code>curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number) : Void</code>	((controlX,controlY)で指定されたコントロールポイントを使用し、現在の描画位置から(anchorX,anchorY)まで、現在の線のスタイルで曲線を描画します。

オプション	シグネチャ	説明
	<code>duplicateMovieClip (name:String, depth:Number, [initObject:Object]): MovieClip</code>	SWF ファイルの再生中に、指定されたムービークリップのインスタンスを作成します。
	<code>endFill():Void</code>	<code>beginFill()</code> メソッドまたは <code>beginGradientFill()</code> メソッドの最後の呼び出し以降に追加された線と曲線に塗りを適用します。
	<code>getBounds(bounds:Object):Object</code>	<i>bounds</i> パラメータに基づいて、ムービークリップの最小および最大の x 座標と y 座標を示すプロパティを返します。
	<code>getBytesLoaded():Number</code>	ムービークリップについて、既にロード(ストリーミング)されたバイト数を返します。
	<code>getBytesTotal():Number</code>	ムービークリップのサイズをバイト単位で返します。
	<code>getDepth():Number</code>	ムービークリップインスタンスの深度を返します。
	<code>getInstanceAtDepth (depth:Number):MovieClip</code>	特定の深度にムービークリップが既に配置されているかどうかを判別します。
	<code>getNextHighest Depth():Number</code>	現在のムービークリップ内の同一レベル、同一レイヤー上の他のすべてのオブジェクトよりもムービークリップを前面に表示する場合に、 <code>MovieClip.attachMovie()</code> メソッド、 <code>MovieClip.duplicateMovieClip()</code> メソッド、 <code>MovieClip.createEmptyMovieClip()</code> メソッドに渡す深度値を調べることができます。
	<code>getSWFVersion():Number</code>	ムービークリップがパブリッシュされたときの対象の Flash Player のバージョンを示す整数値を返します。
	<code>getURL(url:String, [window:String], [method:String]): Void</code>	指定された URL から、指定されたウィンドウにドキュメントを読み込みます。
	<code>globalToLocal (pt:Object):Void</code>	<i>pt</i> オブジェクトをステージ(グローバル)座標からムービークリップ内(ローカル)の座標に変換します。
	<code>gotoAndPlay(frame: Object):Void</code>	指定されたフレームで SWF ファイルの再生を開始します。
	<code>gotoAndStop(frame: Object):Void</code>	このムービークリップの指定されたフレームに再生ヘッドを送り、そこで停止させます。

オプション	シグネチャ	説明
	<code>hitTest() : Boolean</code>	ムービークリップを評価して、それが target、または x 座標と y 座標のパラメータで示されるヒット領域と重なっている（または交差している）かどうかを確認します。
	<code>lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number) : Void</code>	<code>lineTo()</code> メソッドおよび <code>curveTo()</code> メソッドで今後の呼び出しに使用する線のスタイルを指定します。このスタイルは、次に <code>lineStyle()</code> メソッドを異なるパラメータで呼び出すまで使用されます。
	<code>lineTo(x:Number, y:Number) : Void</code>	現在の描画位置から (x, y) まで、現在の線のスタイルを使用して線を描画します。その後で、現在の描画位置は (x, y) に設定されます。
	<code>loadMovie(url:String, [method:String]) : Void</code>	元の SWF ファイルの再生中に、SWF または JPEG ファイルを Flash Player のムービークリップ内にロードします。
	<code>loadVariables(url:String, [method:String]) : Void</code>	外部ファイルからデータを読み取り、ムービークリップの変数の値を設定します。
	<code>localToGlobal(pt:Object) : Void</code>	<code>pt</code> オブジェクトをムービークリップ内（ローカル）の座標からステージ（グローバル）の座標に変換します。
	<code>moveTo(x:Number, y:Number) : Void</code>	現在の描画位置を (x, y) に移動します。
	<code>nextFrame() : Void</code>	次のフレームに再生ヘッドを送り、停止します。
	<code>play() : Void</code>	ムービークリップのタイムラインで再生ヘッドを移動します。
	<code>prevFrame() : Void</code>	直前のフレームに再生ヘッドを戻し、停止します。
	<code>removeMovieClip() : Void</code>	<code>duplicateMovieClip()</code> 、 <code>MovieClip.duplicateMovieClip()</code> <code>MovieClip.createEmptyMovieClip()</code> または <code>MovieClip.attachMovie()</code> で作成したムービークリップインスタンスを削除します。
	<code>setMask(mc:Object) : Void</code>	パラメータ <code>mc</code> のムービークリップをマスクにして、呼び出し元のムービークリップを表示します。

オプション	シグネチャ	説明
	<code>startDrag([lockCenter: Boolean], [left:Number], [top:Number], [right:Number], [bottom:Number]) : Void</code>	指定されたムービークリップをユーザーがドラッグできるようにします。
	<code>stop() : Void</code>	再生中のムービークリップを停止します。
	<code>stopDrag() : Void</code>	<code>MovieClip.startDrag()</code> メソッドを終了します。
	<code>swapDepths(target:Object) : Void</code>	ムービークリップのスタッキング順序、つまり深度(z順序)を、target パラメータに指定したムービーと入れ替えます。または、target パラメータに指定した深度に現在置かれているムービーと入れ替えます。
	<code>unloadMovie() : Void</code>	ムービークリップインスタンスの内容を削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

_alpha (MovieClip._alpha プロパティ)

public _alpha : Number

ムービークリップのアルファ透明度。有効な値は 0(完全な透明)～100(完全な不透明)です。デフォルト値は 100 です。ムービークリップ内で _alpha が 0 に設定されているオブジェクトは、非表示の場合でもアクティブです。たとえば、_alpha プロパティが 0 に設定されているムービークリップ内のボタンをクリックできます。ボタンを完全に無効にするには、ムービークリップの _visible プロパティを false に設定します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次のコードは、マウスがムービークリップ上を移動するときに動的に作成されるムービークリップ triangle の _alpha プロパティを 50% に設定します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());  
  
triangle.beginFill(0x0000FF, 100);  
triangle.moveTo(10, 10);  
triangle.lineTo(10, 100);  
triangle.lineTo(100, 10);  
triangle.lineTo(10, 10);  
  
triangle.onRollOver = function() {  
    this._alpha = 50;  
};  
triangle.onRollOut = function() {  
    this._alpha = 100;  
};
```

関連項目

[_alpha \(Button._alpha プロパティ\)](#), [_alpha \(TextField._alpha プロパティ\)](#),
[_visible \(MovieClip._visible プロパティ\)](#)

attachMovie (MovieClip.attachMovie メソッド)

public attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip

ライブラリからシンボルを取得し、ムービークリップに割り当てます。attachMovie() メソッドで割り当てた SWF ファイルを削除するには、MovieClip.removeMovieClip() または MovieClip.unloadMovie() を使用します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

id:String - ステージ上のムービークリップに割り当てるライブラリ内のムービークリップシンボルのリンクエージ名。これは、[リンクエージプロパティ] ダイアログボックス内の [識別子] フィールドに入力した名前です。

name:String - ムービークリップに割り当てられる一意のインスタンス名。

depth:Number - SWF ファイルが配置される深度を指定する整数。

`initObject:Object`(オプション)-新しく割り当てられたムービークリップに設定するプロパティを含むオブジェクト。このパラメータは、Flash Player 6 以降でのみサポートされています。このパラメータを使用すると、動的に作成したムービークリップは、クリップのパラメータを受け取ることができます。オブジェクトではない `initObject` は無視されます。`initObject` のすべてのプロパティが新しいインスタンスにコピーされます。`initObject` で指定されたプロパティは、コンストラクタ関数で使用できます。

戻り値

`MovieClip`-新しく作成したインスタンスへの参照。

例

次の例では、リンクエージ識別子が "circle" であるシンボルを SWF ファイルのステージ上にあるムービークリップインスタンスに割り当てます。

```
this.attachMovie("circle", "circle1_mc", this.getNextHighestDepth());
this.attachMovie("circle", "circle2_mc", this.getNextHighestDepth(), {_x:100,
_y:100});
```

関連項目

`removeMovieClip` (`MovieClip.removeMovieClip` メソッド), `unloadMovie` (`MovieClip.unloadMovie` メソッド), `removeMovieClip` 関数

beginFill (MovieClip.beginFill メソッド)

`public beginFill(rgb:Number, [alpha:Number]) : Void`

新しい描画パスの始点を示します。開いたパスがあり(現在の描画位置が `MovieClip.moveTo()` メソッドで指定した前の座標と等しくない場合)、そのパスに関連付けられた塗りがあるときは、パスが線で閉じられ、塗りが適用されます。これは、`MovieClip.endFill()` メソッドを呼び出した場合の結果と似ています。

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`rgb:Number`-色を表す 16 進値。たとえば、赤は 0xFF0000、青は 0x0000FF などです。この値を指定しないか、値が `undefined` である場合、塗りは作成されません。

`alpha:Number`(オプション)-塗りのアルファ値を指定する 0 ~ 100 の整数。この値を指定しないと、100(不透明)が使用されます。0 未満の値を指定すると 0 が適用されます。100 を超える値を指定すると 100 が適用されます。

例

次の例では、ステージ上に赤の塗りで四角形を作成します。

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();

www.adobe.com/go/learn\_fl\_samples\_jp の "ActionScript" サンプルフォルダにある "drawingapi.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。
```

関連項目

[moveTo \(MovieClip.moveTo メソッド\)](#), [endFill \(MovieClip.endFill メソッド\)](#),
[beginGradientFill \(MovieClip.beginGradientFill メソッド\)](#)

beginGradientFill (MovieClip.beginGradientFill メソッド)

```
public beginGradientFill(fillType:String, colors:Array, alphas:Array,
    ratios:Array, matrix:Object) : Void
```

新しい描画パスの始点を示します。最初のパラメータが `undefined` であるか、いずれのパラメータも渡されない場合は、パスに関連する塗りがありません。開いたパスがある場合（現在の描画位置が `MovieClip.moveTo()` メソッドで指定された前の座標と等しくない場合）で、パスに関連する塗りがあるときは、そのパスが線で閉じられた後で塗りが適用されます。この動作は `MovieClip.endFill()` メソッドを呼び出したときと似ています。

このメソッドは、次のいずれかの条件が存在する場合は失敗します。

- `colors`、`alphas`、および `ratios` の各パラメータの項目数が等しくない。
- `fillType` パラメータが "linear" でも "radial" でもない。
- `matrix` パラメータのオブジェクトのフィールドのいずれかが無いか、無効である。

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`fillType: String` - ストリング "linear" またはストリング "radial"。

`colors: Array` - グラデーションで使用する RGB 16 進カラー値（赤 `0xFF0000`、青 `0x0000FF` など）の配列。

alphas:Array - colors 配列内の各色に対応するアルファ値の配列。有効な値は 0 ~ 100 です。0 未満の場合は 0 が、100 を超える値の場合は 100 が適用されます。

ratios:Array - 色分布比率の配列。有効な値は 0 ~ 255 です。この値は、100% でサンプリングされる色の幅の割合をパーセントで定義します。

matrix:Object - 次のいずれかの組み合わせのプロパティを持つ変換マトリックスのオブジェクト。

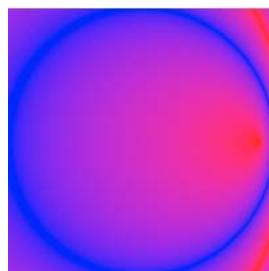
- *a*、*b*、*c*、*d*、*e*、*f*、*g*、*h*、*i* を使用して、3×3 のマトリックスを次の形式で記述できます。

```
a b c  
d e f  
g h i
```

次の例では、このタイプの matrix パラメータを指定した beginGradientFill() メソッドを使用します。

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());  
with (gradient_mc)  
{  
    colors = [0xFF0000, 0x0000FF];  
    fillType = "radial"  
    alphas = [100, 100];  
    ratios = [0, 0xFF];  
    spreadMethod = "reflect";  
    interpolationMethod = "linearRGB";  
    focalPointRatio = 0.9;  
    matrix = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};  
    beginGradientFill(fillType, colors, alphas, ratios, matrix, spreadMethod,  
    interpolationMethod, focalPointRatio);  
    moveTo(100, 100);  
    lineTo(100, 300);  
    lineTo(300, 300);  
    lineTo(300, 100);  
    lineTo(100, 100);  
    endFill();  
}
```

このコードにより、次のイメージが画面に描画されます。



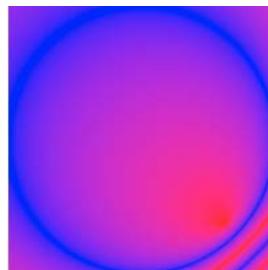
■ matrixType, x, y, w, h, r.

各プロパティの内容は次のとおりです。matrixType はストリング "box" です。x はグラデーションの左上隅の、親クリップの基準点からの相対的な水平座標、y はグラデーションの左上隅の、親クリップの基準点からの相対的な垂直座標を示します。また、w はグラデーションの幅、h はグラデーションの高さ、r はグラデーションのラジアン単位の回転です。

次の例では、このタイプの matrix パラメータを指定した beginGradientFill() メソッドを使用します。

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());
with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial";
    alphas = [100, 100];
    ratios = [0, 0xFF];
    spreadMethod = "reflect";
    interpolationMethod = "linearRGB";
    focalPointRatio = 0.9;
    matrix = {matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
180)*Math.PI};
    beginGradientFill(fillType, colors, alphas, ratios, matrix,
        spreadMethod, interpolationMethod, focalPointRatio);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

このコードにより、次のイメージが画面に描画されます。



関連項目

[beginFill \(MovieClip.beginFill メソッド\)](#), [endFill \(MovieClip.endFill メソッド\)](#),
[lineStyle \(MovieClip.lineStyle メソッド\)](#), [lineTo \(MovieClip.lineTo メソッド\)](#),
[moveTo \(MovieClip.moveTo メソッド\)](#)

clear (MovieClip.clear メソッド)

```
public clear() : Void
```

MovieClip.lineStyle() で指定された線スタイルを含め、ムービークリップの描画メソッドを使用して実行時に作成されたすべてのグラフィックを削除します。Flash の描画ツールを使用してオーリング時に手動で作成したシェイプや線は削除されません。

例

次の例では、ステージ上にボックスを描画します。ユーザーがボックスのグラフィックをクリックすると、ステージ上からグラフィックが削除されます。

```
this.createEmptyMovieClip("box_mc", this.getNextHighestDepth());
box_mc.onRelease = function() {
    this.clear();
};
drawBox(box_mc, 10, 10, 320, 240);
function drawBox(mc:MovieClip, x:Number, y:Number, w:Number, h:Number):Void {
    mc.lineStyle(0);
    mc.beginFill(0xEEEEEE);
    mc.moveTo(x, y);
    mc.lineTo(x+w, y);
    mc.lineTo(x+w, y+h);
    mc.lineTo(x, y+h);
    mc.lineTo(x, y);
    mc.endFill();
}
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "drawingapi.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[lineStyle \(MovieClip.lineStyle メソッド \)](#)

createEmptyMovieClip

(MovieClip.createEmptyMovieClip メソッド)

```
public createEmptyMovieClip(name:String, depth:Number) : MovieClip
```

既存のムービークリップの子として空のムービークリップを作成します。このメソッドの動作は attachMovie() メソッドに似ていますが、新しいムービークリップのリンクエージ識別子を指定する必要はありません。新しく作成された空のムービークリップの基準点は左上隅です。このメソッドは、いずれかのパラメータを省略すると失敗します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`name:String` - 新しいムービークリップのインスタンス名を指定するストリング。

`depth:Number` - 新しいムービークリップの深度を指定する整数。

戻り値

`MovieClip` - 新しく生成されたムービークリップへの参照。

例

次の例では、`container` という名前で空の `MovieClip` を作成します。その中に新しい `TextField` を作成し、新しい `TextField.text` プロパティを設定します。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getNextHighestDepth());
var label:TextField = container.createTextField("label", 1, 0, 0, 150, 20);
label.text = "Hello World";
```

関連項目

`attachMovie (MovieClip.attachMovie メソッド)`

createTextField (MovieClip.createTextField メソッド)

```
public createTextField(instanceName:string, depth:Number, x:Number, y:Number,
    width:Number, height:Number) : TextField
```

このメソッドを呼び出したムービークリップの子となる空のテキストフィールドを新しく作成します。`createTextField()` メソッドを使用すると、ムービーの再生中にテキストフィールドを作成できます。`depth` パラメータは、ムービークリップ内の新しいテキストフィールドの深度 (z 順序の位置) を決定します。それぞれの深度にオブジェクトを 1つだけ含むことができます。既にテキストフィールドが存在する深度に新しいテキストフィールドを作成すると、既存のテキストフィールドが新しいテキストフィールドで置き換えられます。既存のテキストフィールドが上書きされないようにするには、`MovieClip.getInstanceAtDepth()` を使用して特定の深度が既に占有されているかどうかを確認するか、`MovieClip.getNextHighestDepth()` を使用して占有されていない最も上の深度を調べます。テキストフィールドの位置は (`x`, `y`)、幅は `width`、高さは `height` になります。`x` パラメータと `y` パラメータは、コンテナのムービークリップを基準とします。これらのパラメータは、テキストフィールドの `_x` プロパティと `_y` プロパティに対応します。`width` パラメータと `height` パラメータは、テキストフィールドの `_width` プロパティと `_height` プロパティにそれぞれ対応しています。

テキストフィールドのデフォルトのプロパティは次のとおりです。

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
selectable = true
wordWrap = false
mouseWheelEnabled = true
condenseWhite = false
restrict = null
variable = null
maxChars = null
styleSheet = undefined
tabInded = undefined
```

`createTextField()`で作成したテキストフィールドは、次のデフォルトの `TextFormat` オブジェクトの設定を受け取ります。

```
font = "Times New Roman" // "Times" on Mac OS
size = 12
color = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
blockIndent = 0
bullet = false
display = block
tabStops = [] // (empty array)
```

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`instanceName:String` - 新しいテキストフィールドのインスタンス名を識別するストリング。

`depth:Number` - 新しいテキストフィールドの深度を指定する正の整数。

`x:Number` - 新しいテキストフィールドの x 座標を指定する整数。

`y:Number` - 新しいテキストフィールドの y 座標を指定する整数。

`width:Number` - 新しいテキストフィールドの幅を指定する正の整数。

`height:Number` - 新しいテキストフィールドの高さを指定する正の整数。

戻り値

`TextField` -

例

次の例では、幅 300、高さ 100、x 座標 100、y 座標 100、境界なし、赤、および下線付きのテキストフィールドを作成します。

```
this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
var my_fmt:TextFormat = new TextFormat();
my_fmt.color = 0xFF0000;
my_fmt.underline = true;
my_txt.text = "This is my first test field object text.";
my_txt.setTextFormat(my_fmt);
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "animations.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

`getInstanceAtDepth (MovieClip.getInstanceAtDepth メソッド)`,

`getNextHighestDepth (MovieClip.getNextHighestDepth メソッド), TextFormat`

_currentframe (MovieClip._currentframe プロパティ)

`public _currentframe : Number` (読み取り専用)

ムービークリップのタイムライン内で再生ヘッドがあるフレーム番号を返します。

例

次の例では、`_currentframe` プロパティを使用して、ムービークリップ `actionClip_mc` の再生ヘッドを現在の位置から 5 つ先のフレームに進めます。

```
actionClip_mc.gotoAndStop(actionClip_mc._currentframe + 5);
```

curveTo (MovieClip.curveTo メソッド)

```
public curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number)
    : Void
```

((controlX,controlY)で指定されたコントロールポイントを使用し、現在の描画位置から(anchorX,anchorY)まで、現在の線のスタイルで曲線を描画します。その後、現在の描画位置は(anchorX,anchorY)に設定されます。描画先のムービークリップに Flash の描画ツールで作成したコンテンツが含まれている場合、curveTo() の呼び出しの結果はこのコンテンツの下に描画されます。moveTo() メソッドを呼び出す前に curveTo() を呼び出すと、現在の描画位置がデフォルトの(0,0)に設定されます。いずれかのパラメータを省略すると、このメソッドは失敗し、現在の描画位置は変更されません。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

controlX:Number - コントロールポイントの、親ムービークリップの基準点からの相対的な水平座標を指定する整数。

controlY:Number - コントロールポイントの、親ムービークリップの基準点からの相対的な垂直座標を指定する整数。

anchorX:Number - 次のアンカーポイントの、親ムービークリップの基準点からの相対的な水平座標を指定する整数。

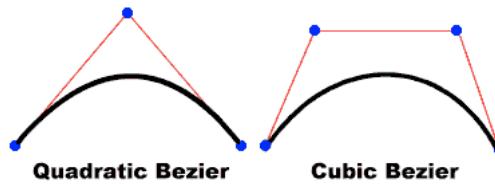
anchorY:Number - 次のアンカーポイントの、親ムービークリップの基準点からの相対的な垂直座標を指定する整数。

例

次の例では、極細の青の実線と赤の塗りで環状に近い曲線を描画します。

```
this.createEmptyMovieClip("circle_mc", 1);
with (circle_mc) {
    lineStyle(0, 0x0000FF, 100);
    beginFill(0xFF0000);
    moveTo(0, 100);
    curveTo(0,200,100,200);
    curveTo(200,200,200,100);
    curveTo(200,0,100,0);
    curveTo(0,0,0,100);
    endFill();
}
```

この例で描画される曲線は、二次ベジエ曲線です。二次ベジエ曲線は、2つのアンカーポイントと1つのコントロールポイントで構成されています。曲線は、2つのアンカーポイントを補間し、コントロールポイントに向かいます。



次のスクリプトでは、`curveTo()` メソッドと `Math` クラスを使用して円を作成します。

```
this.createEmptyMovieClip("circle2_mc", 2);
circle2_mc.lineStyle(0, 0x000000);
drawCircle(circle2_mc, 100, 100, 100);
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, '+y');
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "drawingapi.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

`beginFill` (`MovieClip.beginFill` メソッド), `createEmptyMovieClip` (`MovieClip.createEmptyMovieClip` メソッド), `endFill` (`MovieClip.endFill` メソッド), `lineStyle` (`MovieClip.lineStyle` メソッド), `lineTo` (`MovieClip.lineTo` メソッド), `moveTo` (`MovieClip.moveTo` メソッド), `Math`

_droptarget (MovieClip._droptarget プロパティ)

public _droptarget : String (読み取り専用)

このムービークリップがドロップされたムービークリップインスタンスの絶対パスを、スラッシュシンタックス表記で返します。`_droptarget` プロパティはスラッシュ(/)で始まるパスを常に返します。インスタンスの`_droptarget` プロパティと参照とを比較するには、`eval()` 関数を使用して戻り値をスラッシュシンタックスからドットシンタックス参照に変換します。ActionScript 2.0 ではスラッシュシンタックスをサポートしていません。

メモ: このプロパティは、`System.capabilities.hasMouse` が `true` である、または `System.capabilities.hasStylus` が `true` の場合のみ Flash Lite でサポートされます。

例

次の例では、ムービークリップインスタンス `garbage_mc` の `_droptarget` プロパティを評価し、`eval()` を使用して、評価結果をスラッシュシンタックスからドットシンタックス参照に変換します。次に、`garbage_mc` 参照とムービークリップインスタンス `trashcan_mc` への参照とを比較します。この 2 つの参照が等しい場合、`garbage_mc` の可視性を `false` に設定します。これらの参照が等しくない場合は、`garbage` インスタンスが元の位置にリセットされます。

```
origX = garbage_mc._x;
origY = garbage_mc._y;
garbage_mc.onPress = function() {
    this.startDrag();
};
garbage_mc.onRelease = function() {
    this.stopDrag();
    if (eval(this._droptarget) == trashcan_mc) {
        this._visible = false;
    } else {
        this._x = origX;
        this._y = origY;
    }
};
```

関連項目

[startDrag \(MovieClip.startDrag メソッド\)](#), [stopDrag \(MovieClip.stopDrag メソッド\)](#), [eval 関数](#)

duplicateMovieClip (MovieClip.duplicateMovieClip メソッド)

```
public duplicateMovieClip(name:String, depth:Number, [initObject:Object]) : MovieClip
```

SWF ファイルの再生中に、指定されたムービークリップのインスタンスを作成します。

duplicateMovieClip() メソッドを呼び出したときに元のムービークリップがどのフレーム上にあっても、複製されたムービークリップは常にフレーム 1 から始まります。親のムービークリップ内の変数は、複製されたムービークリップにコピーされません。duplicateMovieClip() メソッドを使用して作成されたムービークリップは、その親で duplicateMovieClip() メソッドを呼び出した場合は複製されません。親のムービークリップが削除されると、複製されたムービークリップも削除されます。MovieClip.loadMovie() または MovieClipLoader クラスを使用してムービークリップをロードした場合、SWF ファイルの内容は複製されません。つまり、JPEG ファイル、GIF ファイル、PNG ファイル、または SWF ファイルをロードした後でムービークリップを複製して帯域幅を節約することはできません。

このメソッドを duplicateMovieClip() のグローバル関数バージョンと比較してみます。このメソッドのグローバル関数バージョンでは、複製するターゲットムービークリップを指定するパラメータが必要になります。MovieClip クラスのバージョンでは、このようなパラメータは不要です。メソッドのターゲットが、メソッドが呼び出されるムービークリップインスタンスであるためです。さらに、duplicateMovieClip() のグローバルバージョンでは、initObject パラメータも、新しく作成された MovieClip インスタンスへの参照の戻り値もサポートされていません。

パラメータ

name: String - 複製したムービークリップの一意の識別子。

depth: Number - 新しいムービークリップを配置する深度を指定する一意の整数。新しいムービークリップインスタンスをオーサリング環境で作成されたすべてのコンテンツの下に配置するには、深度 -16384 を使用します。-16383 以上 -1 以下の値は、オーサリング環境での使用のために予約されているので、このメソッドでは使用しないでください。深度の有効値の範囲は 0 以上 1048575 以下です。

initObject:Object (オプション) - Flash Player 6 以降でのみサポートされます。複製ムービークリップに設定するプロパティを持つオブジェクトを指定します。このパラメータを使用すると、動的に作成したムービークリップは、クリップのパラメータを受け取ることができます。オブジェクトではない initObject は無視されます。initObject のすべてのプロパティが新しいインスタンスにコピーされます。initObject で指定されたプロパティは、コンストラクタ関数で使用できます。

戻り値

MovieClip - 複製されたムービークリップへの参照。Flash Player 6 以降でのみサポートされます。

例

次の例では、新しく作成されたムービークリップを何回か複製し、複製のたびにターゲットをトレースします。

```
var container:MovieClip = setUpContainer();
var ln:Number = 10;
var spacer:Number = 1;
var duplicate:MovieClip;
for(var i:Number = 1; i < ln; i++) {
    var newY:Number = i * (container._height + spacer);
    duplicate = container.duplicateMovieClip("clip-" + i, i, {_y:newY});
    trace(duplicate); // _level0.clip-[number]
}

function setUpContainer():MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip("container",
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 20;
    mc.beginFill(0x333333);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    return mc;
}
```

関連項目

[loadMovie \(MovieClip.loadMovie メソッド\)](#), [removeMovieClip \(MovieClip.removeMovieClip メソッド\)](#), [duplicateMovieClip 関数](#)

enabled (MovieClip.enabled プロパティ)

public enabled : Boolean

ムービークリップが有効であるか無効であるかを示す布尔値。enabled のデフォルト値は true です。enabled を false に設定すると、ムービークリップのコールバックメソッドと onaction イベントハンドラは呼び出されなくなり、Over、Down、Up の各フレームは無効になります。enabled プロパティは、ムービークリップのタイムラインには影響しません。ムービークリップが再生中である場合は、再生が継続されます。ムービークリップは、ムービークリップイベント (mouseDown、mouseUp、keyDown、keyUp など) を引き続き受け取ります。

`enabled` プロパティは、ムービークリップのボタンに似たプロパティのみを制御します。`enabled` プロパティはいつでも変更できます。このプロパティの変更後、ムービークリップはすぐに有効/無効になります。`enabled` プロパティは、プロトタイプオブジェクトから読み取ることができます。`enabled` プロパティが `false` に設定されている場合、オブジェクトは自動タブ順に含まれません。

例

次の例では、ユーザーがムービークリップ `circle_mc` をクリックすると、このムービークリップが無効になります。

```
circle_mc.onRelease = function() {  
    trace("disabling the "+this._name+" movie clip.");  
    this.enabled = false;  
};
```

endFill (MovieClip.endFill メソッド)

public endFill() : Void

`beginFill()` メソッドまたは `beginGradientFill()` メソッドの最後の呼び出し以降に追加された線と曲線に塗りを適用します。適用される塗りは、`beginFill()` または `beginGradientFill()` の前回の呼び出しで指定されたものです。現在の描画位置が `moveTo()` メソッドの直前の呼び出しで指定された座標と等しくない場合、塗りが定義されていれば、パスが線で閉じられた後、塗りが適用されます。

例

次の例では、ステージ上に赤の塗りで四角形を作成します。

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());  
square_mc.beginFill(0xFF0000);  
square_mc.moveTo(10, 10);  
square_mc.lineTo(100, 10);  
square_mc.lineTo(100, 100);  
square_mc.lineTo(10, 100);  
square_mc.lineTo(10, 10);  
square_mc.endFill();
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "drawingapi.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[beginFill \(MovieClip.beginFill メソッド\)](#), [beginGradientFill \(MovieClip.beginGradientFill メソッド\)](#), [moveTo \(MovieClip.moveTo メソッド\)](#)

focusEnabled (MovieClip.focusEnabled プロパティ)

```
public focusEnabled : Boolean
```

このプロパティが `undefined` または `false` の場合、ボタン以外のムービークリップは入力フォーカスを受け取ることができません。`focusEnabled` プロパティの値が `true` の場合は、ムービークリップがボタンでなくともフォーカスを受け取ることができます。

例

次の例では、ムービークリップ `my_mc` の `focusEnabled` プロパティを `false` に設定します。

```
my_mc.focusEnabled = false;
```

_focusrect (MovieClip._focusrect プロパティ)

```
public _focusrect : Boolean
```

ムービークリップに入力フォーカスがあるときに、その周囲に黄色の矩形を表示するかどうかを指定する布尔値。このプロパティは、グローバル `_focusrect` プロパティの設定を上書きできます。ムービークリップインスタンスの `_focusrect` プロパティのデフォルト値は `null` です。つまり、ムービークリップインスタンスは、グローバル `_focusrect` プロパティを上書きしません。ムービークリップインスタンスの `_focusrect` プロパティを `true` または `false` に設定した場合、その1つのムービークリップインスタンスのグローバル `_focusrect` プロパティの設定が上書きされます。

メモ : Flash Lite 2.0 では、`_focusrect` プロパティが無効になっている (`MovieClip._focusrect` が `false` に設定されている) 場合でも、ムービークリップはすべてのキー操作およびマウスイベントを受け取ります。

また、Flash Lite 2.0 では、`fscommand2 SetFocusRectColor` コマンドを使用して、フォーカス矩形のカラーを変更できます。このビヘイビアは、フォーカス矩形のカラーが黄色に制限されている Flash Player とは異なります。

例

この例では、ブラウザウィンドウで、SWF ファイル内の指定されたムービークリップインスタンスにフォーカスがある場合に、その周囲に表示される黄色い矩形を非表示にする方法を示します。`mc1_mc`、`mc2_mc`、`mc3_mc` の3つのムービークリップを作成し、タイムラインのフレーム1に次の ActionScript を追加します。

```
mc1_mc._focusrect = true;
mc2_mc._focusrect = false;
mc3_mc._focusrect = true;

mc1_mc.onRelease = traceOnRelease;
mc3_mc.onRelease = traceOnRelease;

function traceOnRelease() {
    trace(this._name);
}
```

[ファイル]-[パブリッシュプレビュー]-[HTML] を選択して、ブラウザウィンドウで SWF ファイルをテストします。ブラウザウィンドウで SWF をクリックしてフォーカスを与えた後、Tab キーを押して各インスタンスにフォーカスを与えます。`_focusrect` が無効になっている場合は、Enter キーまたはスペースバーを押してブラウザのムービークリップのコードを実行することはできません。

また SWF ファイルのテストはテスト環境で行うことができます。テスト環境では、[制御]-[キーボードショートカットを無効] を選択してください。これにより、SWF ファイルのインスタンスの周囲にあるフォーカスの矩形を表示できます。

関連項目

[_focusrect プロパティ](#), [_focusrect \(Button._focusrect プロパティ\)](#)

`_framesloaded (MovieClip._framesloaded プロパティ)`

public `_framesloaded` : `Number` (読み取り専用)

ストリーミング SWF ファイルからロードされたフレーム数。このプロパティは、特定のフレームとその前のすべてのフレームの内容がロードされており、ユーザーのブラウザでローカルに使用できるかどうかを判別する場合に使用できます。また、大きい SWF ファイルのダウンロードを監視する場合にも便利です。たとえば、SWF ファイルの指定されたフレームがロードを完了するまで、その SWF ファイルがロード中であることを示すメッセージをユーザーに表示する場合に使用できます。

例

次の例では、`_framesloaded` プロパティを使用して、すべてのフレームがロードされた時点で SWF ファイルを開始します。すべてのフレームのロードが完了するまでは、ムービークリップインスタンス `bar_mc` の `_xscale` プロパティは比例的に増加し、ロードの進行状況を示すプログレスバーを作成します。

タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var pctLoaded:Number = Math.round(this.getBytesLoaded() /  
    this.getBytesTotal()*100);  
bar_mc._xscale = pctLoaded;
```

フレーム 2 に次のコードを追加します。

```
if (this._framesloaded < this._totalframes) {  
    this.gotoAndPlay(1);  
} else {  
    this.gotoAndStop(3);  
}
```

フレーム 3 以後にコンテンツを配置し、フレーム 3 に次のコードを追加します。

```
stop();
```

関連項目

[MovieClipLoader](#)

getBounds (MovieClip.getBounds メソッド)

```
public getBounds(bounds:Object) : Object
```

bounds パラメータに基づいて、ムービークリップの最小および最大の x 座標と y 座標を示すプロパティを返します。

メモ : ムービークリップのローカル座標をステージ座標に変換するには、

MovieClip.localToGlobal() メソッドを使用します。ステージ座標をローカル座標に変換するには、MovieClip.globalToLocal() メソッドを使用します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

bounds:Object - その座標系を基準点として使用するタイムラインのターゲットパス。

戻り値

Object - xMin、xMax、yMin、yMax の各プロパティを持つオブジェクトです。

例

次の例では、ムービークリップ square_mc を作成します。このコードはこのムービークリップ用に四角形を描画し、MovieClip.getBounds() を使用してインスタンスの座標値を [出力] パネルに表示します。

```
this.createEmptyMovieClip("square_mc", 1);
square_mc._x = 10;
square_mc._y = 10;
square_mc.beginFill(0xFF0000);
square_mc.moveTo(0, 0);
square_mc.lineTo(100, 0);
square_mc.lineTo(100, 100);
square_mc.lineTo(0, 100);
square_mc.lineTo(0, 0);
square_mc.endFill();

var bounds_obj:Object = square_mc.getBounds(this);
for (var i in bounds_obj) {
    trace(i+" --> "+bounds_obj[i]);
}
```

[出力] パネルに次の情報が表示されます。

```
yMax --> 110
yMin --> 10
xMax --> 110
xMin --> 10
```

関連項目

[globalToLocal \(MovieClip.globalToLocal メソッド\)](#),
[localToGlobal \(MovieClip.localToGlobal メソッド\)](#)

getBytesLoaded (MovieClip.getBytesLoaded メソッド)

public getBytesLoaded():Number

ムービークリップについて、既にロード(ストリーミング)されたバイト数を返します。この値と、MovieClip.getBytesTotal()から返される値を比較して、ロード済みのムービークリップの割合を判別することができます。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

戻り値

Number - ロードされたバイト数を示す整数。

例

次の例では、_framesloaded プロパティを使用して、すべてのフレームがロードされた時点で SWF ファイルを開始します。すべてのフレームのロードが完了するまでは、ムービークリップインスタンス loader の _xscale プロパティは比例的に増加し、ロードの進行状況を示すプログレスバーを作成します。

タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal() * 100);
bar_mc._xscale = pctLoaded;
```

フレーム 2 に次のコードを追加します。

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

フレーム 3 以後にコンテンツを配置し、フレーム 3 に次のコードを追加します。

```
stop();
```

関連項目

[getBytesTotal \(MovieClip.getBytesTotal メソッド\)](#)

getBytesTotal (MovieClip.getBytesTotal メソッド)

public getBytesTotal() : Number

ムービークリップのサイズをバイト単位で返します。外部にあるムービークリップ(ターゲットまたはレベルにロードされるルート SWF ファイルまたはムービークリップ) の場合、戻り値は SWF ファイルの圧縮されていないサイズです。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

戻り値

Number - ムービークリップの総バイト数を示す整数。

例

次の例では、_framesloaded プロパティを使用して、すべてのフレームがロードされた時点で SWF ファイルを開始します。すべてのフレームのロードが完了するまでは、ムービークリップインスタンス loader の _xscale プロパティは比例的に増加し、ロードの進行状況を示すプログレスバーを作成します。

タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var pctLoaded:Number = Math.round(this.getBytesLoaded() /  
    this.getBytesTotal()*100);  
bar_mc._xscale = pctLoaded;
```

フレーム 2 に次のコードを追加します。

```
if (this._framesloaded<this._totalframes) {  
    this.gotoAndPlay(1);  
} else {  
    this.gotoAndStop(3);  
}
```

フレーム 3 以後にコンテンツを配置し、フレーム 3 に次のコードを追加します。

```
stop();
```

関連項目

[getBytesLoaded \(MovieClip.getBytesLoaded メソッド\)](#)

getDepth (MovieClip.getDepth メソッド)

public getDepth() : Number

ムービークリップインスタンスの深度を返します。

各ムービークリップ、ボタン、およびテキストフィールドには関連付けられた一意の深度があり、どのように他のオブジェクトの手前や背後に表示されるかが決まります。深度の高い方のオブジェクトが手前に表示されます。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

戻り値

Number - ムービークリップの深度。

例

次のコードは、ステージ上のすべてのムービークリップインスタンスの深度をトレースします。

```
for (var i in this) {
    if (typeof (this[i]) == "movieclip") {
        trace("movie clip '"+this[i]._name+"' is at depth "+this[i].getDepth());
    }
}
```

関連項目

[getInstanceAtDepth \(MovieClip.getInstanceAtDepth メソッド\)](#),
[getNextHighestDepth \(MovieClip.getNextHighestDepth メソッド\)](#), [swapDepths \(MovieClip.swapDepths メソッド\)](#), [getDepth \(TextField.getDepth メソッド\)](#),
[getDepth \(Button.getDepth メソッド\)](#)

getInstanceAtDepth (MovieClip.getInstanceAtDepth メソッド)

public getInstanceAtDepth(depth:Number) : MovieClip

特定の深度にムービークリップが既に配置されているかどうかを判別します。

MovieClip.attachMovie() メソッド、MovieClip.duplicateMovieClip() メソッド、MovieClip.createEmptyMovieClip() メソッドを使用する前にこのメソッドを使用して、これらのメソッドに渡す深度パラメータに既にムービークリップが置かれているかどうかを調べることができます。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

depth: Number - 照会する深度レベルを指定する整数。

戻り値

MovieClip - 指定した深度に位置する MovieClip インスタンスへの参照。その深度にムービークリップが存在しない場合は undefined。

例

次の例では、ムービークリップインスタンス triangle が配置されている深度を [出力] パネルに表示します。

```
this.createEmptyMovieClip("triangle", 1);

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

trace(this.getInstanceAtDepth(1)); // output: _level0.triangle
```

関連項目

[attachMovie \(MovieClip.attachMovie メソッド\)](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド\)](#), [getDepth \(MovieClip.getDepth メソッド\)](#), [getNextHighestDepth \(MovieClip.getNextHighestDepth メソッド\)](#), [swapDepths \(MovieClip.swapDepths メソッド\)](#)

getNextHighestDepth

(MovieClip.getNextHighestDepth メソッド)

public getNextHighestDepth() : Number

現在のムービークリップ内の同一レベル、同一レイヤー上の他のすべてのオブジェクトよりもムービークリップを前面に表示する場合に、MovieClip.attachMovie() メソッド、MovieClip.duplicateMovieClip() メソッド、MovieClip.createEmptyMovieClip() メソッドに渡す深度値を調べることができます。返される値は 0 以上です。負の値は返されません。サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

メモ : V2 コンポーネントも使用している場合は、このメソッドは使用しないでください。V2 コンポーネントをステージまたはライブラリに配置すると、`getNextHighestDepth()` メソッドは有効範囲外にある深度 1048676 を返します。この結果、`MovieClip.removeMovieClip()` を正常に呼び出せなくなる可能性があります。

戻り値

Number - ムービークリップ内で、同一レベル、同一レイヤー上に存在する他のすべてのムービークリップの前面に表示できる最初の深度インデックスを示す整数。

例

次の例では、`createEmptyMovieClip()` メソッドの `depth` パラメータとして `getNextHighestDepth()` メソッドを使用して、3 つのムービークリップインスタンスを描画します。各ムービークリップに、その深度のラベルを付けます。

```
for (i = 0; i < 3; i++) {
    drawClip(i);
}

function drawClip(n:Number):Void {
    this.createEmptyMovieClip("triangle" + n, this.getNextHighestDepth());
    var mc:MovieClip = eval("triangle" + n);
    mc.beginFill(0x00aaff, 100);
    mc.lineStyle(4, 0xFF0000, 100);
    mc.moveTo(0, 0);
    mc.lineTo(100, 100);
    mc.lineTo(0, 100);
    mc.lineTo(0, 0);
    mc._x = n * 30;
    mc._y = n * 50
    mc.createTextField("label", this.getNextHighestDepth(), 20, 50, 200, 200)
    mc.label.text = mc.getDepth();
}
```

関連項目

[getDepth \(MovieClip.getDepth メソッド\)](#), [getInstanceAtDepth \(MovieClip.getInstanceAtDepth メソッド\)](#), [swapDepths \(MovieClip.swapDepths メソッド\)](#), [attachMovie \(MovieClip.attachMovie メソッド\)](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド\)](#)

getSWFVersion (MovieClip.getSWFVersion メソッド)

```
public getSWFVersion() : Number
```

ムービークリップがパブリッシュされたときの対象の Flash Player のバージョンを示す整数値を返します。ムービークリップが JPEG ファイル、GIF ファイル、PNG ファイルのいずれかの場合、またはエラーが発生してムービークリップの SWF バージョンを判別できなかった場合は、-1 を返します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

戻り値

Number - ムービークリップにロードされている SWF ファイルがパブリッシュされたときに対象となった Flash Player のバージョンを示す整数。

例

次の例では、新しいコンテナを作成し、getSWFVersion() の値を出力します。次に、MovieClipLoader を使用して、Flash Player 7 にパブリッシュされた外部 SWF ファイルをロードし、onLoadInit ハンドラが呼び出された後に getSWFVersion() の値を出力します。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getUpperEmptyDepth());
var listener:Object = new Object();
listener.onLoadInit = function(target:MovieClip):Void {
    trace("target: " + target.getSWFVersion()); // target: 7
}
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(listener);
trace("container: " + container.getSWFVersion()); // container: 8
mcLoader.loadClip("FlashPlayer7.swf", container);
```

getURL (MovieClip.getURL メソッド)

```
public getURL(url:String, [window:String], [method:String]) : Void
```

指定された URL から、指定されたウィンドウにドキュメントを読み込みます。getURL() メソッドでは、GET メソッドまたは POST メソッドを使用して、URL で定義されている別のアプリケーションに変数を渡すこともできます。

Flash コンテンツをホストする Web ページでは、allowScriptAccess 属性を明示的に設定して、Flash Player のスクリプトを HTML コードから許可または拒否する必要があります。この HTML コードは、Internet Explorer では PARAM タグに、Netscape Navigator では EMBED タグにあります。

- allowScriptAccess が "never" の場合、送信スクリプトは常に失敗します。
- allowScriptAccess が "always" の場合、送信スクリプトは常に成功します。

- `allowScriptAccess` が "sameDomain" (バージョン 8 以降の SWF ファイルでサポート) の場合、SWF ファイルがホスト側 Web ページと同じドメインに存在すれば、送信スクリプトが許可されます。
- HTML ページで `allowScriptAccess` が指定されていない場合、デフォルト値は、バージョン 8 の SWF ファイルでは "sameDomain"、バージョン 8 より前の SWF ファイルでは "always" です。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`url:String` - ドキュメントを取得するための URL。

`window:String` (オプション) - ドキュメントのロード先のウィンドウまたは HTML フレームを指定する名前、フレーム、または式。次の予約済みターゲット名のいずれかを使用することもできます。`_self` は現在のウィンドウ内の現在のフレームを指定します。`_blank` は新しいウィンドウを指定します。`_parent` は現在のフレームの親を指定します。`_top` は現在のウィンドウ内のトップレベルのフレームを指定します。

`method:String` (オプション) - ロード対象の SWF ファイルに関連付けられた変数を送信するためのメソッドを指定するストリング。"GET" または "POST" のいずれかを指定します。変数が存在しない場合は、このパラメータを省略します。それ以外の場合は、変数のロードに GET メソッドと POST メソッドのどちらを使用するかを指定してください。GET を指定すると、変数が URL の最後に付加されます。このメソッドは、変数の数が少ない場合に使用します。POST を指定すると、変数が別の HTTP ヘッダーで送信されます。このメソッドは、個々の変数のストリングが長い場合に使用します。

例

次の ActionScript は、新しいムービークリップインスタンスを作成し、新しいブラウザウィンドウで Macromedia Web サイトを開きます。

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.getURL("http://www.macromedia.com", "_blank");
```

`getURL()` メソッドでは、次のようにリモートのサーバー側のスクリプトに変数を送信することもできます。

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.username = "some user input";
loader_mc.password = "random string";
loader_mc.getURL("http://www.flash-mx.com/mm/viewscope.cfm", "_blank", "GET");
```

関連項目

`getURL` 関数, `sendAndLoad` (`LoadVars.sendAndLoad` メソッド), `send` (`LoadVars.send` メソッド)

globalToLocal (MovieClip.globalToLocal メソッド)

```
public globalToLocal(pt:Object) : Void
```

pt オブジェクトをステージ(グローバル)座標からムービークリップ内(ローカル)の座標に変換します。

MovieClip.globalToLocal() メソッドを使用すると、特定の x 座標と y 座標を、ステージの左上隅を基準にした相対値から特定のムービークリップの左上隅を基準にした相対値に変換できます。

最初に x と y の 2 つのプロパティを持つ汎用オブジェクトを作成する必要があります。これらの x 値と y 値(x および y と呼ぶ必要があります)は、ステージの左上隅を基準にしているため、グローバル座標と呼ばれます。x プロパティは、左上隅からの水平オフセットを表します。つまり、そのポイントが基準点からどれだけ右に配置されているかを示します。たとえば、x=50 の場合、そのポイントは左上隅から 50 ピクセル右に配置されていることになります。y プロパティは、左上隅からの垂直オフセットを表します。つまり、そのポイントが基準点からどれだけ下に配置されているかを示します。たとえば、y=20 の場合、そのポイントは左上隅から 20 ピクセル下に配置されていることになります。次のコードでは、これらの座標を持つ汎用オブジェクトを作成します。

```
var myPoint:Object = new Object();
myPoint.x = 50;
myPoint.y = 20;
```

代わりに、オブジェクトを作成し、同時にリテラル Object 値を使用して値を割り当てることもできます。

```
var myPoint:Object = {x:50, y:20};
```

グローバル座標を使用して point オブジェクトを作成したら、その座標をローカル座標に変換できます。globalToLocal() メソッドでは、パラメータとして送る汎用オブジェクトの x 値と y 値を変更するため、値を返しません。このメソッドは、ステージ(グローバル座標)を基準にした値から、特定のムービークリップ(ローカル座標)を基準にした値に変更します。

たとえば、ムービークリップを作成してポイント (_x:100, _y:100) に配置し、ステージの左上隅(x:0, y:0) を表すグローバルポイントを globalToLocal() メソッドに渡すと、このメソッドはその x 値と y 値をローカル座標、この場合(x:-100, y:-100) に変換します。この変換が行われるのは、x 座標と y 座標がこの時点で、ステージの左上隅でなくムービークリップの左上隅を基準にして表現されているためです。値が負になるのは、ムービークリップの左上隅からステージの左上隅に到達するのに、100 ピクセル左(負の x) および 100 ピクセル上(負の y) に移動する必要があるためです。

ムービークリップの座標は、MovieClip の x 値と y 値を設定する MovieClip のプロパティである _x と _y を使用して表現されていました。ただし、汎用オブジェクトでは、アンダースコアなしの x と y を使用します。次のコードでは、x 値と y 値をローカル座標に変換します。

```
var myPoint:Object = {x:0, y:0}; // Create your generic point object.
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());
myMovieClip._x = 100; // _x for movieclip x position
myMovieClip._y = 100; // _y for movieclip y position
```

```
myMovieClip.globalToLocal(myPoint);
trace ("x: " + myPoint.x); // output: -100
trace ("y: " + myPoint.y); // output: -100
```

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`pt:Object` - 汎用の Object クラスを使用して作成したオブジェクトの名前または識別子。このオブジェクトのプロパティで x 座標と y 座標を指定します。

例

次の ActionScript を、イメージ "photo1.jpg" と同じディレクトリ内にある FLA ファイルまたは AS ファイルに追加します。

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
coords_txt.html = true;
coords_txt.multiline = true;
coords_txt.autoSize = true;
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc._x = 100;
target_mc._y = 100;
target_mc.loadMovie("photo1.jpg");

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:_xmouse, y:_ymouse};
    target_mc.globalToLocal(point);
    var rowHeaders = "<b> &ampnbsp \t</b><b>_x\ t</b><b>_y</b>";
    var row_1 = "_root\t"+_xmouse+"\t"+_ymouse;
    var row_2 = "target_mc\t"+point.x+"\t"+point.y;
    coords_txt.htmlText = "<textformat tabstops='[100, 150]'>";
    coords_txt.htmlText += rowHeaders;
    coords_txt.htmlText += row_1;
    coords_txt.htmlText += row_2;
    coords_txt.htmlText += "</textformat>";
};
Mouse.addListener(mouseListener);
```

関連項目

[getBounds \(MovieClip.getBounds メソッド\)](#), [localToGlobal \(MovieClip.localToGlobal メソッド\)](#), [Object](#)

gotoAndPlay (MovieClip.gotoAndPlay メソッド)

```
public gotoAndPlay(frame:Object) : Void
```

指定されたフレームで SWF ファイルの再生を開始します。フレーム以外にシーンも指定するには、`gotoAndPlay()` を使用します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`frame:Object` - 再生ヘッドの送り先となるフレーム番号を表す数値、または再生ヘッドの送り先となるフレームのラベルを表すストリング。

例

次の例では、`_framesloaded` プロパティを使用して、すべてのフレームがロードされた時点で SWF ファイルを開始します。すべてのフレームのロードが完了するまでは、ムービークリップインスタンス `loader` の `_xscale` プロパティは比例的に増加し、ロードの進行状況を示すプログレスバーを作成します。

タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var pctLoaded:Number = Math.round(this.getBytesLoaded() /  
    this.getBytesTotal()*100);  
bar_mc._xscale = pctLoaded;
```

フレーム 2 に次のコードを追加します。

```
if (this._framesloaded<this._totalframes) {  
    this.gotoAndPlay(1);  
} else {  
    this.gotoAndStop(3);  
}
```

フレーム 3 以後にコンテンツを配置し、フレーム 3 に次のコードを追加します。

```
stop();
```

関連項目

[gotoAndPlay 関数](#), [play 関数](#)

gotoAndStop (MovieClip.gotoAndStop メソッド)

```
public gotoAndStop(frame:Object) : Void
```

このムービークリップの指定されたフレームに再生ヘッドを送り、そこで停止させます。フレーム以外にシーンも指定するには、`gotoAndStop()` メソッドを使用します。

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`frame:Object` - 再生ヘッドを送る先のフレーム番号。

例

次の例では、`_framesloaded` プロパティを使用して、すべてのフレームがロードされた時点で SWF ファイルを開始します。すべてのフレームのロードが完了するまでは、ムービークリップインスタンス `loader` の `_xscale` プロパティは比例的に増加し、ロードの進行状況を示すプログレスバーを作成します。

タイムラインのフレーム 1 に次の ActionScript を入力します。

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
    this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

フレーム 2 に次のコードを追加します。

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

フレーム 3 以後にコンテンツを配置し、フレーム 3 に次のコードを追加します。

```
stop();
```

関連項目

[gotoAndStop 関数](#), [stop 関数](#)

_height (MovieClip._height プロパティ)

public _height : Number

ピクセル単位で示したムービークリップの高さです。

例

次のコード例では、ムービークリップの高さと幅を [出力] パネルに表示します。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var image_mc1:MovieClipLoader = new MovieClipLoader();
var mc1Listener:Object = new Object();
mc1Listener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._name+" = "+target_mc._width+" X "+target_mc._height+
        " pixels");
};
image_mc1.addListener(mc1Listener);

image_mc1.loadClip("example.jpg", image_mc);
```

関連項目

[_width \(MovieClip._width プロパティ \)](#)

_highquality (MovieClip._highquality プロパティ)

public _highquality : Number

非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに MovieClip._quality を使用します。

現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。ビットマップスミージングを常にオンにして最高品質を適用するには、2(最高品質)を指定します。アンチエイリアス処理を適用するには、1(高品質)を指定します。SWF ファイルにアニメーションが含まれない場合、ビットマップは滑らかになります。アンチエイリアス処理を避けるには、0(低品質)を指定します。このプロパティでグローバル _highquality プロパティの設定を上書きできます。

例

次の ActionScript は、SWF ファイルに対して最高品質のアンチエイリアス処理の適用を指定します。

```
my_mc._highquality = 2;
```

関連項目

[_quality \(MovieClip._quality プロパティ \), _quality プロパティ](#)

hitArea (MovieClip.hitArea プロパティ)

```
public hitArea : Object
```

ムービークリップのヒット領域となる別のムービークリップを指定します。hitArea プロパティが存在しないか、このプロパティが null または undefined の場合は、ムービークリップ自体がヒット領域として使用されます。hitArea プロパティの値は、ムービークリップオブジェクトへの参照である場合があります。

hitArea プロパティはいつでも変更できます。このプロパティを変更したムービークリップには新しいヒット領域の動作が直ちに反映されます。ヒット領域として指定したムービークリップは可視状態である必要はありません。不可視状態であっても、そのグラフィカルシェイプはヒットテスト可能です。hitArea プロパティは、プロトタイプオブジェクトから読み取ることができます。

例

次の例では、ムービークリップ circle_mc をムービークリップ square_mc のヒット領域として設定します。これらの 2 つのムービークリップをステージ上に配置し、ドキュメントをテストします。ユーザーが circle_mc をクリックすると、ムービークリップ square_mc は、クリックされたことをトレースします。

```
square_mc.hitArea = circle_mc;
square_mc.onRelease = function() {
    trace("hit! "+this._name);
};
```

ムービークリップ circle_mc の visible プロパティを false に設定して、square_mc のヒット領域を非表示にすることもできます。

```
circle_mc._visible = false;
```

関連項目

[hitTest \(MovieClip.hitTest メソッド \)](#)

hitTest (MovieClip.hitTest メソッド)

```
public hitTest() : Boolean
```

ムービークリップを評価して、それが target、または x 座標と y 座標のパラメータで示されるヒット領域と重なっている（または交差している）かどうかを確認します。

シンタックス 1: shapeFlag の設定に従って、x 座標と y 座標を、指定されたインスタンスのシェイプまたは境界ボックスと比較します。shapeFlag を true に設定すると、ステージ上でインスタンスが実際に占有する領域のみが評価されます。x と y が任意の点で重なる場合は、true が返されます。この評価は、ムービークリップが、指定されたヒット領域またはホットスポット領域内にあるかどうかを判別する場合に有効です。

シンタックス 2: target の境界ボックスと指定されたインスタンスの境界ボックスを評価し、両者が任意の点で重なるか交差する場合に true を返します。

パラメータ x: Number はステージ上のヒット領域の x 座標、y: Number は、ステージ上のヒット領域の y 座標です。x 座標と y 座標は、グローバル座標空間で定義されます。また、shapeFlag: Boolean は、指定したインスタンスのシェイプ全体を評価するか (true)、境界ボックスだけを評価するか (false) を指定するブール値です。このパラメータは、x 座標と y 座標のパラメータでヒット領域を指定する場合にのみ指定できます。target: Object は、ムービークリップと交差するか重なるヒット領域のターゲットパスです。target パラメータは通常、ボタンまたはテキスト入力フィールドを表します。

戻り値

Boolean - ブール値。指定したヒット領域にムービークリップが重なっている(交差している)場合は true、それ以外の場合は false を返します。

例

次の例では、hitTest() を使用して、ユーザーがマウスボタンを離したときにムービークリップ circle_mc がムービークリップ square_mc と重なっている(交差している)かどうかを判別します。

```
square_mc.onPress = function() {
    this.startDrag();
};

square_mc.onRelease = function() {
    this.stopDrag();
    if (this.hitTest(circle_mc)) {
        trace("you hit the circle");
    }
};
```

関連項目

[getBounds \(MovieClip.getBounds メソッド\)](#), [globalToLocal \(MovieClip.globalToLocal メソッド\)](#), [localToGlobal \(MovieClip.localToGlobal メソッド\)](#)

lineStyle (MovieClip.lineStyle メソッド)

```
public lineStyle(thickness:Number, rgb:Number, alpha:Number,
    pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String,
    miterLimit:Number) : Void
```

lineTo() メソッドおよび curveTo() メソッドで今後の呼び出しに使用する線のスタイルを指定します。このスタイルは、次に lineStyle() メソッドを異なるパラメータで呼び出すまで使用されます。パスの描画中に lineStyle() メソッドを呼び出し、パス内の線のセグメントごとに異なるスタイルを指定できます。

メモ: clear() を呼び出すと、線のスタイルが undefined に戻ります。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

thickness:Number - 線の太さをポイント単位で示す整数。有効な値は 0 ~ 255 です。数値を指定しない場合、または undefined を指定した場合、線は描画されません。0 未満の値を指定した場合は 0 が使用されます。0 は極細線です。最大の太さは 255 です。255 を超える値を指定した場合は 255 が使用されます。

rgb:Number - 線の色を表す 16 進値。たとえば、赤は 0xFF0000、青は 0x0000FF で表します。値を指定しないと、0x000000(黒)が使用されます。

alpha:Number - 線の色のアルファ値を示す整数。有効な値は 0 ~ 100 です。値を指定しないと 100(ソリッド)が使用されます。0 未満の値を指定すると 0 が適用されます。100 を超える値を指定すると 100 が適用されます。

pixelHinting:Boolean -

noScale:String -

capsStyle:String -

jointStyle:String -

miterLimit:Number -

例

次の例では、5 ピクセルのマゼンタ色の実線を使用して、塗りなしの三角形を描画します。

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.lineStyle(5, 0xff00ff, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
```

関連項目

[beginFill \(MovieClip.beginFill メソッド\)](#), [beginGradientFill \(MovieClip.beginGradientFill メソッド\)](#), [clear \(MovieClip.clear メソッド\)](#), [curveTo \(MovieClip.curveTo メソッド\)](#), [lineTo \(MovieClip.lineTo メソッド\)](#), [moveTo \(MovieClip.moveTo メソッド\)](#)

lineTo (MovieClip.lineTo メソッド)

```
public lineTo(x:Number, y:Number) : Void
```

現在の描画位置から (x, y) まで、現在の線のスタイルを使用して線を描画します。その後で、現在の描画位置は (x, y) に設定されます。描画先のムービークリップに Flash の描画ツールで作成したコンテンツが含まれている場合、`lineTo()` の呼び出しの結果はこのコンテンツの下に描画されます。`moveTo()` メソッドを呼び出す前に `lineTo()` メソッドを呼び出すと、現在の描画位置はデフォルトの $(0,0)$ になります。いずれかのパラメータを省略すると、このメソッドは失敗し、現在の描画位置は変更されません。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`x:Number` - 親ムービークリップの基準点からの相対的な水平座標を示す整数。

`y:Number` - 親ムービークリップの基準点からの相対的な垂直座標を示す整数。

例

次の例では、5 ピクセルのマゼンタ色の実線と、部分的に透明な青の塗りを使用して三角形を描画します。

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

関連項目

[beginFill \(MovieClip.beginFill メソッド \)](#),
[createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド \)](#),
[endFill \(MovieClip.endFill メソッド\)](#), [lineStyle \(MovieClip.lineStyle メソッド \)](#),
[moveTo \(MovieClip.moveTo メソッド \)](#)

loadMovie (MovieClip.loadMovie メソッド)

```
public loadMovie(url:String, [method:String]) : Void
```

元の SWF ファイルの再生中に、SWF または JPEG ファイルを Flash Player のムービークリップ内にロードします。

ヒント : ダウンロードの進捗状況を監視するには、loadMovie() メソッドではなく、MovieClipLoader.loadClip() メソッドを使用します。

loadMovie() メソッドを使用しない場合は、Flash Player が 1 つの SWF ファイルを表示して終了します。loadMovie() メソッドを使用すると、複数の SWF ファイルを同時に表示し、別の HTML ドキュメントをロードせずに SWF ファイルを切り替えることができます。

ムービークリップ内にロードした SWF ファイルまたはイメージは、そのムービークリップの位置、回転、および拡大 / 縮小の各プロパティを継承します。ムービークリップのターゲットパスを使用して、ロードした SWF ファイルをターゲットとして設定できます。

loadMovie() メソッドを呼び出すときは、次のサンプルコードに示すように、ロード先のムービー内で MovieClip._lockroot プロパティを true に設定します。ロード先のムービー内で _lockroot を true に設定しない場合、ロードされたムービー内の _root への参照は、ロードされたムービーの _root ではなく、ロード先の _root をポイントします。

```
myMovieClip._lockroot = true;
```

loadMovie() メソッドでロードした SWF ファイルまたはイメージを削除するには、MovieClip.unloadMovie() メソッドを使用します。

アクティブな SWF ファイルを保持して、新しいデータをその中にロードするには、MovieClip.loadVariables() メソッド、XML オブジェクト、Flash Remoting オブジェクトまたは Runtime Shared オブジェクトを使用します。

イベントハンドラと MovieClip.loadMovie() を併用すると、予期しない結果が生じる可能性があります。on() を使用してイベントハンドラをボタンに割り当てるか、MovieClip.onPress のようなイベントハンドラメソッドを使用してダイナミックハンドラを作成した後、loadMovie() を呼び出すと、新しいコンテンツがロードされた後にイベントハンドラが維持されません。一方、onClipEvent() または on() を使用してイベントハンドラをムービークリップに割り当てた後、そのムービークリップで loadMovie() を呼び出すと、新しいコンテンツがロードされた後にイベントハンドラが維持されます。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`url:String` - ロードする SWF ファイルまたは JPEG ファイルの絶対 URL または相対 URL。相対パスは、レベル 0 の SWF ファイルが埋め込まれた HTML ファイルを基準にする必要があります。絶対 URL の場合は `http://` や `file:///` などのプロトコル参照を含めて指定します。

`method:String`(オプション)- 変数を送信またはロードするための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送信する変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないとときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、新しいムービークリップを作成した後、その中に子を作成し、子に PNG イメージをロードします。この場合、`loadMovie` の呼び出し前に親に割り当てられていたインスタンスの値がすべて保持されます。

```
var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.onRelease = function():Void {
    trace(this.image._url); // http://www.w3.org/Icons/w3c_main.png
}
var image:MovieClip = mc.createEmptyMovieClip("image",
    mc.getNextHighestDepth());
image.loadMovie("http://www.w3.org/Icons/w3c_main.png");
```

関連項目

[_lockroot](#) (`MovieClip._lockroot` プロパティ), [unloadMovie](#)
[\(MovieClip.unloadMovie メソッド\)](#), [loadVariables](#) (`MovieClip.loadVariables` メソッド), [loadMovie](#) (`MovieClip.loadMovie` メソッド), [onPress](#) (`MovieClip.onPress` ハンドラ), [MovieClipLoader](#), [onClipEvent](#) ハンドラ, [on](#) ハンドラ, [loadMovieNum](#) 関数, [unloadMovie](#) 関数, [unloadMovieNum](#) 関数

loadVariables (`MovieClip.loadVariables` メソッド)

`public loadVariables(url:String, [method:String]) : Void`

外部ファイルからデータを読み取り、ムービークリップの変数の値を設定します。外部ファイルとして、ColdFusion によって作成されたテキストファイル、CGI スクリプト、ASP (Active Server Page: アクティブサーバーページ)、PHP スクリプトのほか、適切に書式化されたテキストファイルを使用できます。ファイルには任意の数の変数を含むことができます。

また、`loadVariables` メソッドを使用して、アクティブなムービークリップ内の変数の値を更新できます。

`loadVariables` メソッドでは、URL のテキストが標準の MIME 形式の `application/x-www-form-urlencoded` (CGI スクリプト形式) である必要があります。

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、`url` は、呼び出し元の SWF ファイルと同じスープードメインに属している必要があります。スープードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、`www.someDomain.com` に存在する SWF ファイルは、`store.someDomain.com` に存在するデータソースからデータをロードできます。これは、どちらのファイルも同じスープードメイン `someDomain.com` に属しているからです。

Flash Player 7 以降で動作する SWF ファイルでは、`url` に、この呼び出しを発行する SWF ファイルと正確に同じドメインを使用する必要があります。たとえば、`www.someDomain.com` に存在する SWF ファイルは、`www.someDomain.com` に存在するソース以外からはデータをロードできません。異なるドメインからデータをロードするには、アクセス対象のデータソースをホストするサーバーにクロスドメインポリシーファイルを置く必要があります。

変数を特定のレベルにロードするには、`loadVariables()` の代わりに `loadVariablesNum()` を使用します。

サブクラスを作成することにより、`MovieClip` クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

`url:String` - ロードする変数がある外部ファイルの絶対 URL または相対 URL。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、`url` は SWF ファイルと同じドメインに属している必要があります。詳細については、次の「説明」を参照してください。

`method:String` (オプション) - 変数を送信するための HTTP メソッドを指定します。パラメータはストリング GET または POST でなければなりません。送信する変数がない場合は、このパラメータを省略します。GET メソッドは、変数を URL の最後に追加します。このメソッドは、変数のデータ量が少ないときに使用します。POST メソッドは、別の HTTP ヘッダで変数を送信します。このメソッドは、変数のデータ量が多いときに使用します。

例

次の例では、テキストファイル "params.txt" の情報を、`createEmptyMovieClip()` を使用して作成されたムービークリップ `target_mc` にロードします。`setInterval()` 関数は、ロードの進捗状況をチェックする場合に使用します。このスクリプトは、"params.txt" ファイル内の `done` という名前の変数をチェックします。

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc.loadVariables("params.txt");
function checkParamsLoaded() {
    if (target_mc.done == undefined) {
        trace("not yet.");
    }
}
```

```

} else {
    trace("finished loading. killing interval.");
    trace("-----");
    for (i in target_mc) {
        trace(i+": "+target_mc[i]);
    }
    trace("-----");
    clearInterval(param_interval);
}
}

var param_interval = setInterval(checkParamsLoaded, 100);
"params.txt" ファイルには、次のテキストが含まれています。
var1="hello"&var2="goodbye"&done="done"

```

関連項目

[loadMovie \(MovieClip.loadMovie メソッド\)](#), [loadVariablesNum 関数](#), [unloadMovie \(MovieClip.unloadMovie メソッド\)](#)

localToGlobal (MovieClip.localToGlobal メソッド)

public localToGlobal(pt:[Object](#)) : [Void](#)

pt オブジェクトをムービークリップ内(ローカル) の座標からステージ(グローバル) の座標に変換します。

`MovieClip.localToGlobal()` メソッドを使用すると、指定された x 座標と x 座標を、特定のムービークリップの左上隅を基準にした相対値からステージの左上隅を基準にした相対値に変換できます。

最初に x と y の 2 つのプロパティを持つ汎用オブジェクトを作成する必要があります。これらの x 値と y 値(x および y と呼ぶ必要があります)は、ムービークリップの左上隅を基準にしているため、ローカル座標と呼ばれます。 x プロパティは、ムービークリップの左上隅からの水平オフセットを表します。つまり、そのポイントが基準点からどれだけ右に配置されているかを示します。たとえば、 x = 50 の場合、そのポイントは左上隅から 50 ピクセル右に配置されていることになります。 y プロパティは、ムービークリップの左上隅からの垂直オフセットを表します。つまり、そのポイントが基準点からどれだけ下に配置されているかを示します。たとえば、 y = 20 の場合、そのポイントは左上隅から 20 ピクセル下に配置されることになります。次のコードでは、これらの座標を持つ汎用オブジェクトを作成します。

```

var myPoint:Object = new Object();
myPoint.x = 50;
myPoint.y = 20;

```

代わりに、オブジェクトを作成し、同時にリテラル Object 値を使用して値を割り当てることもできます。

```

var myPoint:Object = {x:50, y:20};

```

ローカル座標を使用して point オブジェクトを作成したら、その座標をグローバル座標に変換できます。localToGlobal() メソッドでは、パラメータとして送る汎用オブジェクトの x 値と y 値を変更するため、値を返しません。このメソッドは、特定のムービークリップ(ローカル座標)を基準にした値から、ステージ(グローバル座標)を基準にした値に変更します。

たとえば、ムービークリップを作成してポイント(_x:100,_y:100)に配置し、ムービークリップの左上隅に近いポイント(x:10,y:10)を表すローカルポイントをlocalToGlobal() メソッドに渡す場合、このメソッドでは、その x 値と y 値をグローバル座標、この場合(x:110,y:110)に変換する必要があります。この変換が行われるのは、x 座標と y 座標がこの時点で、ムービークリップの左上隅でなくステージの左上隅を基準にして表現されているためです。

ムービークリップの座標は、MovieClip の x 値と y 値を設定する MovieClip のプロパティである _x と _y を使用して表現されていました。ただし、汎用オブジェクトでは、アンダースコアなしの x と y を使用します。次のコードでは、x 座標と y 座標をグローバル座標に変換します。

```
var myPoint:Object = {x:10, y:10}; // create your generic point object
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());
myMovieClip._x = 100; // _x for movieclip x position
myMovieClip._y = 100; // _y for movieclip y position

myMovieClip.localToGlobal(myPoint);
trace ("x: " + myPoint.x); // output: 110
trace ("y: " + myPoint.y); // output: 110
```

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

pt:Object - x 座標と y 座標をプロパティとして指定し、Object クラスを使用して作成したオブジェクトの名前または識別子。

例

次の例では、オブジェクト my_mc の x 座標と y 座標を、ムービークリップ内の座標(ローカル)からステージ座標(グローバル)に変換します。インスタンスをクリックしてドラッグした後に、ムービークリップの中心点が反映されます。

```
this.createTextField("point_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:my_mc._width/2, y:my_mc._height/2};
    my_mc.localToGlobal(point);
    point_txt.text = "x:"+point.x+", y:"+point.y;
};
Mouse.addListener(mouseListener);
my_mc.onPress = function() {
    this.startDrag();
};
```

```
my_mc.onRelease = function() {
  this.stopDrag();
};
```

関連項目

[globalToLocal \(MovieClip.globalToLocal メソッド\)](#)

_lockroot (MovieClip._lockroot プロパティ)

```
public _lockroot : Boolean
```

SWF ファイルがムービークリップにロードされたときに、_root で参照する内容を指定する布尔値。_lockroot プロパティのデフォルト値は undefined です。このプロパティは、ロードする SWF ファイルで設定することも、ムービークリップをロードするハンドラで設定することもできます。

たとえば、ユーザーがプレイするゲームを選択したら、そのゲーム ("Chess.swf" など) をムービークリップ game_mc にロードする、"Games.fla" という名前のドキュメントがあるとします。

"Games.swf" にロードされた後も、"Chess.swf" で使用されている _root がすべて、"Games.swf" の _root ではなく、"Chess.swf" の _root を参照する必要があります。"Chess.fla" にアクセスでき、それを Flash Player 7 以降にパブリッシュする場合、メインタイムラインでこのステートメントを "Chess.fla" に追加できます。

```
this._lockroot = true;
```

"Chess.fla" にアクセスできない場合 (たとえば、"Chess.swf" を他の人のサイトから chess_mc にロードしている場合)、ロード時に "Chess.swf" の _lockroot プロパティを設定できます。次の ActionScript を "Games.fla" のメインタイムラインに配置します。

```
chess_mc._lockroot = true;
```

この例では、"Games.swf" が Flash Player 7 以降用にパブリッシュされていれば、"Chess.swf" はどのバージョンの Flash Player 用にでもパブリッシュすることができます。

loadMovie() の呼び出しでは、次のコードに示すように、ロード先のムービーで

MovieClip._lockroot プロパティを true に設定します。ロード先のムービー内で _lockroot を true に設定しない場合、ロードされたムービー内の _root への参照は、ロードされたムービーの _root ではなく、ロード先の _root をポイントします。

```
myMovieClip._lockroot = true;
```

例

次の例では、"lockroot.fla" の _lockroot がメインの SWF ファイルに適用されています。SWF ファイルが別の FLA ドキュメントにロードされる場合、_root は常に "lockroot.swf" のスコープを参照します。これにより、コンフリクトを回避できます。次の ActionScript を "lockroot.fla" のメインタイムラインに配置します。

```
this._lockroot = true;
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from lockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

次の情報が表示されます。

```
from lockroot.swf
myOtherVar -> 2
myVar -> 1
_lockroot -> true
$version -> WIN 7,0,19,0
```

次の例では、"lockroot.swf" および "nolockroot.swf" の 2 つの SWF ファイルをロードします。"lockroot.fla" ドキュメントには前の例の ActionScript が含まれています。nolockroot FLA ファイルでは、タイムラインのフレーム 1 に次のコードがあります。

```
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from nolockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

"lockroot.swf" ファイルには _lockroot が適用されていますが、"nolockroot.swf" ファイルには適用されていません。ファイルがロードされた後、各ファイルがそれぞれの _root スコープから変数をダンプします。次の ActionScript を FLA ドキュメントのメインタイムフレームに配置します。

```
this.createEmptyMovieClip("lockroot_mc", this.getNextHighestDepth());
lockroot_mc.loadMovie("lockroot.swf");
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc.loadMovie("nolockroot.swf");
function dumpRoot() {
    trace("from current SWF file");
    for (i in _root) {
        trace(" "+i+" -> "+_root[i]);
    }
    trace("");
}
dumpRoot();
```

次の情報が表示されます。

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myVar -> 1
i -> lockroot_mc
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

_lockroot が適用されていないファイルには、ルート SWF 内の他の変数もすべて含まれます。
"nolockroot.fla" にアクセスできない場合、メインタイムラインに追加した次の ActionScript を使用して、前のメインの FLA ドキュメントの _lockroot を変更できます。

```
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc._lockroot = true;
nolockroot_mc.loadMovie("nolockroot.swf");
```

次のように表示されます。

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myOtherVar -> 2
myVar -> 1

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

関連項目

[_root プロパティ](#), [_lockroot \(MovieClip._lockroot プロパティ\)](#), [attachMovie \(MovieClip.attachMovie メソッド\)](#), [loadMovie \(MovieClip.loadMovie メソッド\)](#), [onLoadInit \(MovieClipLoader.onLoadInit イベントリスナー\)](#)

moveTo (MovieClip.moveTo メソッド)

```
public moveTo(x:Number, y:Number) : Void
```

現在の描画位置を (x, y) に移動します。いずれかのパラメータを省略すると、このメソッドは失敗し、現在の描画位置は変更されません。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

x:Number - 親ムービークリップの基準点からの相対的な水平座標を示す整数。

y:Number - 親ムービークリップの基準点からの相対的な垂直座標を示す整数。

例

次の例では、5 ピクセルのマゼンタ色の実線と、部分的に透明な青の塗りを使用して三角形を描画します。

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

関連項目

[createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド\)](#), [lineStyle \(MovieClip.lineStyle メソッド\)](#), [lineTo \(MovieClip.lineTo メソッド\)](#)

_name (MovieClip._name プロパティ)

```
public _name : String
```

ムービークリップのインスタンス名。

関連項目

[_name \(Button._name プロパティ\)](#)

nextFrame (MovieClip.nextFrame メソッド)

```
public nextFrame() : Void
```

次のフレームに再生ヘッドを送り、停止します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の例では、_framesloaded と nextFrame() を使用して、SWF ファイルにコンテンツをロードします。タイムラインのフレーム 1 にはコードを追加せず、フレーム 2 に次の ActionScript を追加します。

```
if (this._framesloaded >= 3) {  
    this.nextFrame();  
} else {  
    this.gotoAndPlay(1);  
}
```

次に、次のコード（およびロードする必要のあるコンテンツ）をフレーム 3 に追加します。

```
stop();
```

関連項目

[nextFrame 関数](#) , [prevFrame 関数](#) , [prevFrame \(MovieClip.prevFrame メソッド\)](#)

onData (MovieClip.onData ハンドラ)

```
onData = function() {}
```

ムービークリップが MovieClip.loadVariables() の呼び出しありは MovieClip.loadMovie() の呼び出しからデータを受け取ったときに呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

このハンドラは、クラスに関連付けられたシンボルがライブラリに存在するムービークリップでのみ使用できます。特定のムービークリップがデータを受け取ったときにイベントハンドラを呼び出す場合は、このハンドラの代わりに onClipEvent() を使用します。任意のムービークリップがデータを受け取ったときは、後者のイベントハンドラが呼び出されます。

例

次の例は、MovieClip.onData() および onClipEvent(data) の正しい使用方法を示しています。

symbol_mc はライブラリ内のムービークリップシンボルです。このムービークリップシンボルは MovieClip クラスにリンクされます。次の最初の関数は、symbol_mc の各インスタンスがデータを受け取ったときに、そのインスタンスごとにトリガれます。

dynamic_mc は、MovieClip.loadMovie() を使用してロード中のムービークリップです。次に示す dynamic_mc を使用するコードでは、ムービークリップをロードしたときに関数を呼び出そうとしますが、動作しません。ロードされる SWF ファイルは、MovieClip クラスに関連付けられたライブラリ内のシンボルである必要があります。

最後の関数では onClipEvent(data) が使用されます。onClipEvent() イベントハンドラは、ムービークリップがライブラリ内にあるかどうかにかかわらず、データを受け取った任意のムービークリップに対して起動されます。そのため、この例の最後の関数は、symbol_mc がインスタンス化されたときと、"replacement.swf" がロードされたときに起動されます。

```
// The following function is triggered for each instance of symbol_mc
// when it receives data.
symbol_mc.onData = function() {
    trace("The movie clip has received data");
}

// This code attempts to call a function when the clip is loaded,
// but it will not work, because the loaded SWF is not a symbol
// in the library associated with the MovieClip class.
function output()
{
    trace("Will never be called.");
}
dynamic_mc.onData = output;
dynamic_mc.loadMovie("replacement.swf");
// The following function is invoked for any movie clip that
// receives data, whether it is in the library or not.
onClipEvent( data ) {
    trace("The movie clip has received data");
}
```

関連項目

[onClipEvent ハンドラ](#)

onDragOut (MovieClip.onDragOut ハンドラ)

```
onDragOut = function() {}
```

マウスボタンが押された状態で、ピントがオブジェクトの外に移動したときに呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ: このイベントハンドラは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onDragOut イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onDragOut = function () {
    trace ("onDragOut called");
}
```

関連項目

[onDragOver \(MovieClip.onDragOver ハンドラ \)](#)

onDragOver (MovieClip.onDragOver ハンドラ)

```
onDragOver = function() {}
```

ピントがムービークリップ外にドラッグされた後で、ムービークリップ上に移動したときに呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ: このイベントハンドラは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onDragOver イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onDragOver = function () {
    trace ("onDragOver called");
}
```

関連項目

[onDragOut \(MovieClip.onDragOut ハンドラ \)](#)

onEnterFrame (MovieClip.onEnterFrame ハンドラ)

```
onEnterFrame = function() {}
```

SWF ファイルのフレームレートで繰り返し呼び出されます。onEnterFrame イベントハンドラに割り当てる関数は、影響を受けるフレームに追加されている他のすべての ActionScript コードの前に処理されます。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

例

次の例では、onEnterFrame イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onEnterFrame = function () {
    trace ("onEnterFrame called");
}
```

onKeyDown (MovieClip.onKeyDown ハンドラ)

```
onKeyDown = function() {}
```

ムービークリップにフォーカスがあるときにキーを押すと、呼び出されます。onKeyDown イベントハンドラにはパラメータは渡されません。Key.getAscii() メソッドと Key.getCode() メソッドを使用して、どのキーが押されたかを調べることができます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

onKeyDown イベントハンドラは、ムービークリップの入力フォーカスが有効で、設定されている場合にのみ動作します。まず、ムービークリップの MovieClip.focusEnabled プロパティを true に設定します。次に、ムービークリップにフォーカスを与えます。そのためには、Selection.setFocus() を使用するか、Tab キーでムービークリップに移動するように設定します。Selection.setFocus() を使用する場合は、ムービークリップのパスを Selection.setFocus() に渡す必要があります。フォーカスは、マウスを動かすことによって他のエレメントに簡単に移動します。

例

次の例では、onKeyDown() イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。ムービークリップ my_mc を作成し、次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
my_mc.onKeyDown = function () {  
    trace ("key was pressed");  
}
```

onKeyDown イベントハンドラを実行するには、ムービークリップにフォーカスがあることが必要です。次の ActionScript を追加して入力フォーカスを設定します。

```
my_mc.tabEnabled = true;  
my_mc.focusEnabled = true;  
Selection.setFocus(my_mc);
```

Tab キーを使用してムービークリップに移動してキーを押すと、[出力] パネルに key was pressed と表示されます。しかし、マウスを動かした後は、ムービークリップからフォーカスが移動されてしまうため、key was pressed は表示されません。このため、ほとんどの場合は Key.onKeyDown を使用する必要があります。

関連項目

[getAscii \(Key.getAscii メソッド\)](#), [getCode \(Key.getCode メソッド\)](#), [focusEnabled \(MovieClip.focusEnabled プロパティ\)](#), [setFocus \(Selection.setFocus メソッド\)](#), [onKeyDown \(Key.onKeyDown イベントリスナー\)](#), [onKeyUp \(MovieClip.onKeyUp ハンドラ\)](#)

onKeyUp (MovieClip.onKeyUp ハンドラ)

```
onKeyUp = function() {}
```

キーを離すと呼び出されます。onKeyUp イベントハンドラにはパラメータは渡されません。Key.getAscii() メソッドと Key.getCode() メソッドを使用して、どのキーが押されたかを調べることができます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

onKeyUp イベントハンドラは、ムービークリップの入力フォーカスが有効で、設定されている場合にのみ動作します。まず、ムービークリップの MovieClip.focusEnabled プロパティを true に設定します。次に、ムービークリップにフォーカスを与えます。そのためには、Selection.setFocus() を使用するか、Tab キーでムービークリップに移動するように設定します。

Selection.setFocus() を使用する場合は、ムービークリップのパスを Selection.setFocus() に渡す必要があります。フォーカスは、マウスを動かすことによって他のエレメントに簡単に移動します。

例

次の例では、onKeyUp イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onKeyUp = function () {
    trace ("onKey called");
}
```

次の例では、入力フォーカスを設定します。

```
my_mc.focusEnabled = true;
Selection.setFocus(my_mc);
```

関連項目

[getAscii \(Key.getAscii メソッド\)](#), [getCode \(Key.getCode メソッド\)](#), [focusEnabled \(MovieClip.focusEnabled プロパティ\)](#), [setFocus \(Selection.setFocus メソッド\)](#), [onKeyDown \(Key.onKeyDown イベントリスナー\)](#), [onKeyDown \(MovieClip.onKeyDown ハンドラ\)](#)

onKillFocus (MovieClip.onKillFocus ハンドラ)

```
onKillFocus = function(newFocus:Object) {}
```

ムービークリップが入力フォーカスを失うと、呼び出されます。onKillFocus メソッドは、1つのパラメータ newFocus を受け取ります。このパラメータは、フォーカスを受け取る新しいオブジェクトを表すオブジェクトです。フォーカスを受け取るオブジェクトがない場合、newFocus の値は null です。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

パラメータ

newFocus: Object - 入力フォーカスを受け取るオブジェクト。

例

次の例では、フォーカスを失うムービークリップに関する情報と現在フォーカスがあるインスタンスを表示します。ステージ上には、my_mc と other_mc の 2 つのムービークリップがあります。次の ActionScript を AS ドキュメントまたは FLA ドキュメントに追加できます。

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onKillFocus = function(newFocus) {
    trace("onKillFocus called, new focus is: "+newFocus);
};
```

Tab キーを押して 2 つのインスタンス間を移動すると、[出力] パネルに情報が表示されます。

関連項目

[onSetFocus \(MovieClip.onSetFocus ハンドラ\)](#)

onLoad (MovieClip.onLoad ハンドラ)

```
onLoad = function() {}
```

ムービークリップのインスタンスが作成されてタイムラインに読み込まれると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

このハンドラは、クラスに関連付けられたシンボルがライブラリに存在するムービークリップでのみ使用できます。たとえば、MovieClip.loadMovie() を使用して SWF ファイルを動的にロードする場合など、特定のムービークリップがロードされたときにイベントハンドラを呼び出す場合は、このハンドラの代わりに、onClipEvent(load) クラスまたは MovieClipLoader クラスを使用します。他のハンドラは、MovieClip.onLoad とは異なり、どのムービークリップがロードされたときにも呼び出されます。

例

この例は、MovieClip クラスを拡張する ActionScript 2.0 クラス定義における onLoad イベントハンドラの使用方法を示しています。まず、"Oval.as" という名前のクラスファイルを作成し、onLoad() という名前のクラスメソッドを定義します。クラスファイルは適切なクラスパスに配置します。

```
// contents of Oval.as
class Oval extends MovieClip{
    public function onLoad () {
        trace ("onLoad called");
    }
}
```

次に、ライブラリ内にムービークリップシンボルを作成し、Oval という名前を付けます。[ライブラリ] パネルでシンボルをコンテキストクリック(通常右クリック)し、ポップアップメニューから [リンクエージ] を選択します。[ActionScript に書き出し] をクリックし、[識別子] フィールドと [AS 2.0 クラス] フィールドに "Oval" という語を入力します(引用符は含めません)。[最初のフレームに書き出し] をオンのままにして [OK] をクリックします。

次に、ファイルの最初のフレームに移動して、[アクション] パネルで次のコードを入力します。

```
var myOval:Oval = Oval(attachMovie("Oval","Oval_1",1));
```

最後に、ムービープレビューを実行します。"onLoad called" という出力テキストが表示されます。

関連項目

[loadMovie \(MovieClip.loadMovie メソッド\), onClipEvent ハンドラ, MovieClipLoader](#)

onMouseDown (MovieClip.onMouseDown ハンドラ)

```
onMouseDown = function() {}
```

マウスボタンが押されると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ : このイベントハンドラは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onMouseDown イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onMouseDown = function () {
    trace ("onMouseDown called");
}
```

onMouseMove (MovieClip.onMouseMove ハンドラ)

```
onMouseMove = function() {}
```

マウスポインタが移動すると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ : このイベントハンドラは、System.capabilities.hasMouse が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onMouseMove イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onMouseMove = function () {
    trace ("onMouseMove called");
}
```

onMouseUp (MovieClip.onMouseUp ハンドラ)

```
onMouseUp = function() {}
```

マウスボタンが離されると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ: このイベントハンドラは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onMouseUp イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onMouseUp = function () {
    trace ("onMouseUp called");
}
```

onPress (MovieClip.onPress ハンドラ)

```
onPress = function() {}
```

ポイントタガムービークリップ上にあるときにマウスをクリックすると、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はライブラリに定義できます。

例

次の例では、onPress イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onPress = function () {
    trace ("onPress called");
}
```

onRelease (MovieClip.onRelease ハンドラ)

```
onRelease = function() {}
```

ムービークリップの上でマウスボタンを離すと、呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

例

次の例では、onRelease イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onRelease = function () {
    trace ("onRelease called");
}
```

onReleaseOutside (MovieClip.onReleaseOutside ハンドラ)

```
onReleaseOutside = function() {}
```

マウスボタンをムービークリップ領域内で押して、ムービークリップ領域の外側で離したときに呼び出されます。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

メモ: このイベントハンドラは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、onReleaseOutside イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onReleaseOutside = function () {
    trace ("onReleaseOutside called");
}
```

onRollOut (MovieClip.onRollOut ハンドラ)

```
onRollOut = function() {}
```

ピントがムービークリップ領域の外側に移動すると、呼び出されます。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

例

次の例では、onRollOut イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onRollOut = function () {
    trace ("onRollOut called");
}
```

onRollOver (MovieClip.onRollOver ハンドラ)

```
onRollOver = function() {}
```

ピントがムービークリップ領域の上に移動すると、呼び出されます。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

例

次の例では、onRollOver イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onRollOver = function () {
    trace ("onRollOver called");
}
```

onSetFocus (MovieClip.onSetFocus ハンドラ)

```
onSetFocus = function(oldFocus:Object) {}
```

ムービークリップが入力フォーカスを受け取ると、呼び出されます。*oldFocus* パラメータは、フォーカスを失うオブジェクトです。たとえば、ユーザーが Tab キーを押して入力フォーカスをムービークリップからテキストフィールドに移動すると、*oldFocus* にムービークリップインスタンスが入ります。

前にフォーカスがあったオブジェクトが存在しない場合、*oldFocus* には null 値が入ります。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

パラメータ

oldFocus:[Object](#) - フォーカスを失うオブジェクト。

例

次の例では、入力フォーカスを受け取るムービークリップに関する情報と、前にフォーカスがあったインスタンスが表示されます。ステージ上には、my_mc と other_mc の 2 つのムービークリップがあります。次の ActionScript を AS ドキュメントまたは FLA ドキュメントに追加します。

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onSetFocus = function(oldFocus) {
    trace("onSetFocus called, previous focus was: "+oldFocus);
}
```

2つのインスタンス間で Tab キーを押すと、[出力] パネルに情報が表示されます。

関連項目

[onKillFocus \(MovieClip.onKillFocus ハンドラ\)](#)

onUnload (MovieClip.onUnload ハンドラ)

```
onUnload = function() {}
```

ムービークリップをタイムラインから削除した後に表示される最初のフレームで呼び出されます。onUnload イベントハンドラに関連付けられているアクションは、影響を受けるフレームにアクションが割り当てられる前に処理されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。関数はタイムラインに定義できます。また、MovieClip クラスを拡張するクラスファイル内、またはライブラリ内のシンボルにリンクされるクラスファイル内に定義できます。

例

次の例では、MovieClip.onUnload イベントハンドラの関数を定義します。この関数は、[出力] パネルに trace() アクションを送ります。

```
my_mc.onUnload = function () {
    trace ("onUnload called");
}
```

_parent (MovieClip._parent プロパティ)

```
public _parent : MovieClip
```

現在のボタンを含むムービークリップを指す参照です。現在のオブジェクトは、_parent プロパティを参照するオブジェクトです。現在のムービークリップまたはオブジェクトの上位にあるムービークリップまたはオブジェクトへの相対パスを指定するには、_parent プロパティを使用します。

_parent を使用して表示リストの複数のレベルを上に移動するには、次のようにします。

```
this._parent._parent._alpha = 20;
```

例

次の例では、ムービークリップとメインタイムラインとの関係への参照をトレースします。my_mc というインスタンス名でムービークリップを作成し、それをメインタイムラインに追加します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
my_mc.onRelease = function() {
    trace("You clicked the movie clip: "+this);
    trace("The parent of "+this._name+" is: "+this._parent);
}
```

ムービークリップをクリックすると、[出力] パネルに次の情報が表示されます。

```
You clicked the movie clip: _level0.my_mc
The parent of my_mc is: _level0
```

関連項目

[_parent \(Button._parent プロパティ\)](#), [_root プロパティ](#), [targetPath 関数](#), [_parent \(TextField._parent プロパティ\)](#)

play (MovieClip.play メソッド)

```
public play() : Void
```

ムービークリップのタイムラインで再生ヘッドを移動します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の ActionScript を使用して SWF ファイルのメインタイムラインを再生します。これは、メインタイムライン上のムービークリップボタン my_mc 用の ActionScript です。

```
stop();
my_mc.onRelease = function() {
    this._parent.play();
};
```

次の ActionScript を使用して、SWF ファイル内のムービークリップのタイムラインを再生します。これは、ムービークリップ animation_mc を再生する、メインタイムライン上のボタン my_btn の ActionScript です。

```
animation_mc.stop();
my_btn.onRelease = function(){
    animation_mc.play();
};
```

関連項目

[play 関数](#), [gotoAndPlay \(MovieClip.gotoAndPlay メソッド\)](#), [gotoAndPlay 関数](#)

prevFrame (MovieClip.prevFrame メソッド)

public prevFrame() : Void

直前のフレームに再生ヘッドを戻し、停止します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の例では、2つのムービークリップボタンがタイムラインを制御します。prev_mc ボタンは再生ヘッドを前のフレームに移動し、next_mc ボタンは再生ヘッドを次のフレームに移動します。タイムライン上の一連のフレームにコンテンツを追加し、次の ActionScript をタイムラインのフレーム1に追加します。

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
        parent_mc.prevFrame();
    } else {
        parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
        parent_mc.nextFrame();
    } else {
        parent_mc.gotoAndStop(1);
    }
};
```

関連項目

[prevFrame 関数](#)

_quality (MovieClip._quality プロパティ)

public _quality : [String](#)

SWF ファイルに使用するレンダリング品質を設定または取得します。デバイスフォントは常にエイリアス処理されるため、_quality プロパティには影響されません。

_quality プロパティは、次の値に設定できます。

値	説明	グラフィックのアンチエイリアス
"LOW"	低いレンダリング品質。	グラフィックスはアンチエイリアス処理されていません。
"MEDIUM"	普通のレンダリング品質。この設定は、テキストを含まないムービーに適しています。	グラフィックスは 2×2 ピクセルグリッドを使用してアンチエイリアス処理されます。
"HIGH"	高いレンダリング品質。デフォルトのレンダリング品質設定です。	グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理されます。
"BEST"	非常に高いレンダリング品質。	グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理されます。

メモ: このプロパティを MovieClip オブジェクトに対して指定することができますが、実際にはグローバルプロパティであるので、単に _quality という形で値を指定することもできます。

例

この例では、ムービークリップ my_mc のレンダリング品質を LOW に設定します。

```
my_mc._quality = "LOW";
```

関連項目

[_quality プロパティ](#)

removeMovieClip (MovieClip.removeMovieClip メソッド)

```
public removeMovieClip():Void  
duplicateMovieClip(), MovieClip.duplicateMovieClip()  
MovieClip.createEmptyMovieClip() または MovieClip.attachMovie() で作成したムー  
ビークリップインスタンスを削除します。
```

このメソッドは、負の深度値に割り当てられているムービークリップを削除しません。オーサリングツールで作成したムービークリップには、デフォルトで負の深度値が割り当てられます。負の深度値に割り当てられているムービークリップを削除するには、最初に MovieClip.swapDepths() を使用して、ムービークリップを正の深度値に移動します。

メモ: V2 のコンポーネントを使用している場合、DepthManager クラスでなく MovieClip.getNextHighestDepth() を使用して深度値を割り当てるとき、removeMovieClip() はエラー通知なしで失敗します。何らかの V2 コンポーネントを使用している場合、DepthManager クラスでは、カーソルとツールヒントに対して使用可能な最高 (1048575) および最低 (-16383) の深度を自動的に予約します。その後 getNextHighestDepth() を呼び出すと、有効範囲外にある 1048576 が返されます。removeMovieClip() メソッドは、有効範囲外にある深度値を見つけると、エラー通知なしで失敗します。getNextHighestDepth() をバージョン 2 のコンポーネントと併用する必要がある場合、有効な深度値を割り当てるには swapDepths()、ムービークリップのコンテンツを削除するには MovieClip.unloadMovie() を使用できます。代わりに、DepthManager クラスを使用して、有効範囲内の深度値を割り当てることができます。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の例では、ボタンをクリックするたびに、ムービークリップインスタンスをステージのランダムな位置に割り当てます。ムービークリップインスタンスをクリックすると、そのインスタンスが SWF ファイルから削除されます。

```
function randRange(min:Number, max:Number):Number {  
    var randNum:Number = Math.round(Math.random()*(max-min))+min;  
    return randNum;  
}  
var bugNum:Number = 0;  
addBug_btn.onRelease = addBug;  
function addBug() {  
    var thisBug:MovieClip = this._parent.attachMovie("bug_id", "bug"+bugNum+"_mc",  
        bugNum,  
        {_x:randRange(50, 500), _y:randRange(50, 350)});  
    thisBug.onRelease = function() {  
        this.removeMovieClip();  
    };
```

```
    bugNum++;
}
```

関連項目

[duplicateMovieClip 関数](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip メソッド\)](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#), [attachMovie \(MovieClip.attachMovie メソッド\)](#), [swapDepths \(MovieClip.swapDepths メソッド\)](#)

_rotation (MovieClip._rotation プロパティ)

public _rotation : Number

ムービークリップの元の位置からの回転角度を度数で指定します。時計回りに回転させる場合は 0 ~ 180 の値を指定します。反時計回りに回転させる場合は 0 ~ -180 の値を指定します。この範囲を超える値は、360 の倍数を加算または減算され、範囲内に収まる値になるように調整されます。たとえば、my_mc._rotation = 450 というステートメントは my_mc._rotation = 90 と同義です。

例

次の例では、ムービークリップインスタンス triangle を動的に作成します。SWF ファイルを実行し、ムービークリップをクリックすると、ムービークリップが回転します。

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

triangle.onMouseUp= function() {
    this._rotation += 15;
};
```

関連項目

[_rotation \(Button._rotation プロパティ\)](#), [_rotation \(TextField._rotation プロパティ\)](#)

setMask (MovieClip.setMask メソッド)

```
public setMask(mc:Object) : Void
```

パラメータ *mc* のムービークリップをマスクにして、呼び出し元のムービークリップを表示します。

setMask() メソッドを使用すると、複雑な多重レイヤーのコンテンツを持つ複数フレームのムービークリップをマスクとして設定できます（マスクレイヤーを使用すれば可能です）。マスクされたムービークリップにあるデバイスフォントは、描画されますがマスクされません。ムービークリップをそれ自身のマスクとして指定することはできません。たとえば、*my_mc.setMask(my_mc)* とは記述できません。

ムービークリップを含むマスクレイヤーを作成してから、それに *setMask()* メソッドを適用すると、*setMask()* の呼び出しが優先され、元に戻すことはできません。たとえば、マスクレイヤー内のムービークリップ *UIMask* をマスクする別のレイヤーの別のムービークリップ *UIMask* があるとします。SWF ファイルを再生し、*UIMask.setMask(UIMask)* を呼び出すと、その時点から *UIMask* は *UIMask* によってマスクされます。

ActionScript で作成したマスクをキャンセルするには、*setMask()* に *null* を渡します。次のコードでは、タイムラインのマスクレイヤーに影響を与えずにマスクをキャンセルします。

```
UIMask.setMask(null);
```

サブクラスを作成することにより、*MovieClip* クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

mc: Object - マスクとして使用するムービークリップのインスタンス名。ストリングまたは *MovieClip* を使用できます。

例

次の例では、ムービークリップ *circleMask_mc* をムービークリップ *theMaskee_mc* のマスクとして使用します。

```
theMaskee_mc.setMask(circleMask_mc);
```

_soundbuftime (MovieClip._soundbuftime プロパティ)

```
public _soundbuftime : Number
```

サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。

メモ: このプロパティは、*MovieClip* オブジェクトに対して指定できますが、実際にはロードされたすべてのサウンドに適用されるグローバルプロパティであるので、単に *_soundbuftime* として値を指定することもできます。*MovieClip* オブジェクトにこのプロパティを設定すると、実際にはグローバルプロパティが設定されます。

関連項目

[_soundbuftime プロパティ](#)

startDrag (MovieClip.startDrag メソッド)

```
public startDrag([lockCenter:Boolean], [left:Number], [top:Number],  
[right:Number], [bottom:Number]) : Void
```

指定されたムービークリップをユーザーがドラッグできるようにします。MovieClip.stopDrag() を呼び出して明示的に停止するか、他のムービークリップをドラッグ可能にするまでの間、ムービーはドラッグ可能なままになります。一度に1つのムービークリップのみドラッグ可能です。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

メモ: このメソッドは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

パラメータ

lockCenter:Boolean (オプション) - ドラッグ可能なムービークリップがマウス位置の中心にロックされるか(true)、ユーザーがムービークリップ上で最初にクリックした点にロックされるか(false)を指定する布尔値。

left:Number (オプション) - ムービークリップの制限矩形を指定するムービークリップの親の座標を基準にした相対値。

top:Number (オプション) - ムービークリップの制限矩形を指定するムービークリップの親の座標を基準にした相対値。

right:Number (オプション) - ムービークリップの制限矩形を指定するムービークリップの親の座標を基準にした相対値。

bottom:Number (オプション) - ムービークリップの制限矩形を指定するムービークリップの親の座標を基準にした相対値。

例

次の例では、ドラッグ可能なムービークリップインスタンス mc_1 を作成します。

```
this.createEmptyMovieClip("mc_1", 1);
```

```
with (mc_1) {  
    lineStyle(1, 0xCCCCCC);  
    beginFill(0x4827CF);  
    moveTo(0, 0);  
    lineTo(80, 0);  
    lineTo(80, 60);  
    lineTo(0, 60);  
    lineTo(0, 0);  
    endFill();  
}
```

```
mc_1.onPress = function() {
```

```
    this.startDrag();
};

mc_1.onRelease = function() {
    this.stopDrag();
};
```

関連項目

[_droptarget \(MovieClip._droptarget プロパティ\)](#), [startDrag 関数](#),
[stopDrag \(MovieClip.stopDrag メソッド\)](#)

stop (MovieClip.stop メソッド)

public stop() : Void

再生中のムービークリップを停止します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の例では、ムービークリップ aMovieClip を停止します。

```
aMovieClip.stop();
```

関連項目

[stop 関数](#)

stopDrag (MovieClip.stopDrag メソッド)

public stopDrag() : Void

MovieClip.startDrag() メソッドを終了します。このメソッドによってドラッグ可能になったムービークリップは、stopDrag() メソッドを実行するか、他のムービークリップがドラッグ可能になるまで、ドラッグ可能のままでです。一度に1つのムービークリップのみドラッグ可能です。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

メモ: このメソッドは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、ドラッグ可能なムービークリップインスタンス mc_1 を作成します。

```
this.createEmptyMovieClip("mc_1", 1);

with (mc_1) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0x4827CF);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
}

mc_1.onPress = function() {
    this.startDrag();
};
mc_1.onRelease = function() {
    this.stopDrag();
};
```

関連項目

[_droptarget \(MovieClip._droptarget プロパティ\), startDrag \(MovieClip.startDrag メソッド\), stopDrag 関数](#)

swapDepths (MovieClip.swapDepths メソッド)

public swapDepths(target:[Object](#)) : Void

ムービークリップのスタッキング順序、つまり深度 (z 順序) を、target パラメータに指定したムービーと入れ替えます。または、target パラメータに指定した深度に現在置かれているムービーと入れ替えます。両方のムービークリップは、同じ親ムービークリップに属している必要があります。ムービークリップの深度を入れ替えると、あるムービークリップを他のムービークリップの前面または背面に移動させるという効果が得られます。このメソッドを呼び出すときにムービークリップがトゥイーンしている場合、トゥイーンは停止します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

パラメータ

target: Object - このパラメータは、次のいずれかの形式で指定できます。

- ムービークリップを配置する深度を指定する数値。
- 別のムービークリップインスタンスを指定するストリング。指定したムービークリップインスタンスとこのメソッドの適用先ムービークリップの深度が入れ替わります。両方のムービークリップは、同じ親ムービークリップに属している必要があります。

例

次の例では、2つのムービークリップインスタンスの重ね順を入れ替えます。`myMC1_mc` と `myMC2_mc` の2つのムービークリップインスタンスをステージ上で重ねた後、親のタイムラインに次のスクリプトを追加します。

```
myMC1_mc.onRelease = function() {
    this.swapDepths(myMC2_mc);
};

myMC2_mc.onRelease = function() {
    this.swapDepths(myMC1_mc);
};
```

関連項目

[_level プロパティ](#), [getDepth \(MovieClip.getDepth メソッド\)](#), [getInstanceAtDepth \(MovieClip.getInstanceAtDepth メソッド\)](#), [getNextHighestDepth \(MovieClip.getNextHighestDepth メソッド\)](#)

tabChildren (MovieClip.tabChildren プロパティ)

public tabChildren : Boolean

ムービークリップの子が自動タブ順に含まれているかどうかを判別します。`tabChildren` プロパティが `undefined` または `true` の場合、そのムービークリップの子は自動タブ順に含まれます。`tabChildren` の値が `false` の場合、ムービークリップの子は自動タブ順に含まれません。デフォルト値は `undefined` です。

例

ムービークリップとして、複数の項目を含むリストボックスユーザーインターフェイスウィジェットを作成するとします。ユーザーが各項目をクリックして選択できるように、各項目をボタンとします。ただし、Tab キーで移動できるのはリストボックス自体のみにします。リストボックス内の項目はタブ順から除外します。項目をタブ順から除外するには、リストボックスの `tabChildren` プロパティを `false` に設定します。

`tabIndex` プロパティが使用されている場合、`tabChildren` プロパティは意味を持ちません。`tabChildren` プロパティは自動タブ順にのみ影響します。

次の例では、親ムービークリップ menu_mc 内のすべての子ムービークリップのタブ順が無効になります。

```
menu_mc.onRelease = function(){};
menu_mc.menu1_mc.onRelease = function(){};
menu_mc.menu2_mc.onRelease = function(){};
menu_mc.menu3_mc.onRelease = function(){};
menu_mc.menu4_mc.onRelease = function(){};
menu_mc.tabChildren = false;
```

menu_mc の子のムービークリップインスタンスを自動タブ順に含めるには、コードの最後の行を次のように変更します。

```
menu_mc.tabChildren = true;
```

関連項目

[tabIndex \(Button.tabIndex プロパティ\)](#), [tabEnabled \(MovieClip.tabEnabled プロパティ\)](#), [tabIndex \(MovieClip.tabIndex プロパティ\)](#), [tabIndex \(TextField.tabIndex プロパティ\)](#)

tabEnabled (MovieClip.tabEnabled プロパティ)

```
public tabEnabled : Boolean
```

ムービークリップが自動タブ順に含まれるかどうかを示します。デフォルト値は `undefined` です。

`tabEnabled` プロパティが `undefined` の場合、オブジェクトは、そのオブジェクトで少なくとも1つのムービークリップハンドラ (`MovieClip.onRelease` など) が定義されている場合にのみ、自動タブ順に含まれます。`tabEnabled` が `true` の場合、オブジェクトは自動タブ順に含まれます。

`tabIndex` プロパティにも値を設定すると、オブジェクトはカスタムタブ順にも含まれます。

`tabEnabled` が `false` の場合は、`tabIndex` プロパティが設定されていても、オブジェクトは自動タブ順またはカスタムタブ順に含まれません。ただし、`MovieClip.tabChildren` が `true` の場合、`tabEnabled` が `false` に設定されている子を含め、ムービークリップの子を自動タブ順に含むことができます。

例

次の例では、`myMC2_mc` が自動タブ順に含まれません。

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC2_mc.tabEnabled = false;
```

関連項目

[onRelease \(MovieClip.onRelease ハンドラ\)](#), [tabEnabled \(Button.tabEnabled プロパティ\)](#), [tabChildren \(MovieClip.tabChildren プロパティ\)](#), [tabIndex \(MovieClip.tabIndex プロパティ\)](#), [tabEnabled \(TextField.tabEnabled プロパティ\)](#)

tabIndex (MovieClip.tabIndex プロパティ)

public tabIndex : Number

ムービー内のオブジェクトのタブ順をカスタマイズできます。tabIndex プロパティのデフォルト値は undefined です。tabIndex プロパティは、ボタン、ムービークリップ、またはテキストフィールドの各インスタンスで設定できます。

SWF ファイルのオブジェクトに tabIndex プロパティがある場合、自動タブ順は無効になり、タブ順は SWF ファイルのオブジェクトの tabIndex プロパティから計算されます。カスタムタブ順には、tabIndex プロパティを持つオブジェクトのみが含まれます。

tabIndex プロパティは正の整数である必要があります。オブジェクトのタブ順は、その tabIndex プロパティに従って昇順に決定されます。tabIndex の値が 1 であるオブジェクトは、tabIndex の値が 2 であるオブジェクトよりも前になります。カスタムタブ順は、SWF ファイルのオブジェクトの階層関係を無視します。tabIndex プロパティがある SWF ファイルのすべてのオブジェクトは、タブ順に挿入されます。複数のオブジェクトの tabIndex に同じ値を使用しないでください。

例

次の ActionScript は、3 つのムービークリップインスタンスのカスタムタブ順を設定します。

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC1_mc.tabIndex = 2;
myMC2_mc.tabIndex = 1;
myMC3_mc.tabIndex = 3;
```

関連項目

[tabIndex \(Button.tabIndex プロパティ\)](#), [tabIndex \(TextField.tabIndex プロパティ\)](#)

_target (MovieClip._target プロパティ)

public _target : **String** (読み取り専用)

ムービークリップインスタンスのターゲットパスをスラッシュ表記で返します。ターゲットパスをドット表記に変換するには、eval() 関数を使用します。

例

次の例では、SWF ファイル内のムービークリップインスタンスのターゲットパスを、スラッシュ表記とドット表記で表示します。

```
for (var i in this) {
    if (typeof (this[i]) == "movieclip") {
        trace("name: " + this[i]._name + ",\t target: " + this[i]._target + ",\t
        target(2):"
            + eval(this[i]._target));
    }
}
```

_totalframes (MovieClip._totalframes プロパティ)

public _totalframes : **Number** (読み取り専用)

MovieClip パラメータで指定されたムービークリップインスタンスのフレームの総数を返します。

例

次の例では、2つのムービークリップボタンがタイムラインを制御します。prev_mc ボタンは再生ヘッドを前のフレームに移動し、next_mc ボタンは再生ヘッドを次のフレームに移動します。タイムライン上の一連のフレームにコンテンツを追加し、次の ActionScript をタイムラインのフレーム1に追加します。

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
        parent_mc.prevFrame();
    } else {
        parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
        parent_mc.nextFrame();
    } else {
        parent_mc.gotoAndStop(1);
    }
};
```

trackAsMenu (MovieClip.trackAsMenu プロパティ)

```
public trackAsMenu : Boolean
```

他のボタンまたはムービークリップがマウスまたはスタイルスから解放イベントを受け取ることができるかどうかを示す布尔値です。ムービークリップ上でスタイルスまたはマウスをドラッグし、2番目のムービークリップでマウスボタンを離すと、2番目のムービークリップに対して onRelease イベントが登録されます。これにより、2番目のムービークリップのメニューが作成されます。trackAsMenu プロパティは、任意のボタンまたはムービークリップオブジェクトで設定できます。trackAsMenu プロパティを定義していない場合、デフォルトの動作は false です。

trackAsMenu プロパティは必要に応じていつでも変更できます。変更は即座に反映されます。

メモ :このプロパティは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、ステージ上の 3 つのムービークリップの trackAsMenu プロパティを設定します。1つ のムービークリップをクリックし、2番目のムービークリップでマウスボタンを離すと、イベントを受け取るインスタンスを確認できます。

```
myMC1_mc.trackAsMenu = true;
myMC2_mc.trackAsMenu = true;
myMC3_mc.trackAsMenu = false;

myMC1_mc.onRelease = clickMC;
myMC2_mc.onRelease = clickMC;
myMC3_mc.onRelease = clickMC;

function clickMC() {
    trace("you clicked the "+this._name+" movie clip.");
}
```

関連項目

[trackAsMenu \(Button.trackAsMenu プロパティ \)](#)

unloadMovie (MovieClip.unloadMovie メソッド)

```
public unloadMovie() : Void
```

ムービークリップインスタンスの内容を削除します。インスタンスプロパティとクリップハンドラは残ります。

プロパティとクリップハンドラも含めてインスタンスを削除するには、MovieClip.removeMovieClip() を使用します。

サブクラスを作成することにより、MovieClip クラスのメソッドおよびイベントハンドラを拡張できます。

例

次の例では、ユーザーがムービークリップインスタンス box をクリックしたときに、ムービークリップ box をアンロードします。

```
this.createEmptyMovieClip("box", 1);

with (box) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0x4827CF);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
}

box.onRelease = function() {
    box.unloadMovie();
};
```

関連項目

[removeMovieClip \(MovieClip.removeMovieClip メソッド\)](#), [attachMovie \(MovieClip.attachMovie メソッド\)](#), [loadMovie \(MovieClip.loadMovie メソッド\)](#), [unloadMovie 関数](#), [unloadMovieNum 関数](#)

_url (MovieClip._url プロパティ)

public _url : [String](#) (読み取り専用)

ムービークリップのダウンロード元である SWF、JPEG、GIF、または PNG の各ファイルの URL を取得します。

例

次の例では、インスタンス image_mc にロードされたイメージの URL を [出力] パネルに表示します。

```
this.createEmptyMovieClip("image_mc", 1);
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("_url: "+target_mc._url);
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mcListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

_visible (MovieClip._visible プロパティ)

public _visible : Boolean

ムービークリップを表示するかどうかを示す布尔値です。_visible プロパティが `false` に設定されている非表示のムービークリップは、使用できません。たとえば、_visible プロパティが `false` に設定されたムービークリップ内のボタンはクリックできません。

例

次の例では、2つのムービークリップ、`myMC1_mc` および `myMC2_mc` の _visible プロパティを設定します。一方のインスタンスのプロパティを `true` に設定し、もう一方のインスタンスのプロパティを `false` に設定します。インスタンス `myMC1_mc` は、_visible プロパティが `false` に設定された後はクリックできなくなります。

```
myMC1_mc.onRelease = function() {
    trace(this._name + "._visible = false");
    this._visible = false;
};

myMC2_mc.onRelease = function() {
    trace(this._name + ".alpha = 0");
    this._alpha = 0;
};
```

関連項目

[_visible \(Button._visible プロパティ\)](#), [_visible \(TextField._visible プロパティ\)](#)

_width (MovieClip._width プロパティ)

public _width : Number

ピクセル単位で示したムービークリップの幅。

例

次のコード例では、ムービークリップの高さと幅を [出力] パネルに表示します。

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

trace(triangle._name + " = " + triangle._width + " X " + triangle._height + " pixels");
```

関連項目

[_height \(MovieClip._height プロパティ\)](#)

_x (MovieClip._x プロパティ)

public _x : Number

親ムービークリップのローカル座標を基準にしてムービークリップの x 座標を設定する整数。ムービークリップがメインタイムラインにある場合、その座標系はステージの左上隅を (0,0) として参照します。ムービークリップが、変形されている別のムービークリップの内部にある場合、そのムービークリップの座標系は、それを囲む親ムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップの場合、そのムービークリップの子は、反時計回りに 90 度回転した座標系を継承します。ムービークリップの座標は、基準点の位置を参照します。

関連項目

[_xscale \(MovieClip._xscale プロパティ\)](#), [_y \(MovieClip._y プロパティ\)](#), [_yscale \(MovieClip._yscale プロパティ\)](#)

_xmouse (MovieClip._xmouse プロパティ)

public _xmouse : Number (読み取り専用)

マウス位置の x 座標を返します。

メモ: このプロパティは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、ステージ (_level0) 上のマウスの現在の x 座標と y 座標を、ステージ上のムービークリップ my_mc を基準とする相対位置として返します。

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = "&nbsp;\t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
    mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
    mouse_txt.htmlText += row1_str;
    mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
    mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
    mouse_txt.htmlText += "</textformat>";
};
```

関連項目

[Mouse, _ymouse \(MovieClip._ymouse プロパティ\)](#)

_xscale (MovieClip._xscale プロパティ)

public _xscale : Number

ムービークリップの基準点から適用するムービークリップの水平方向の拡大 / 縮小率 (percentage) を設定します。デフォルトの基準点は (0,0) です。

ローカル座標系を拡大 / 縮小すると、_x および _y プロパティの設定に影響します。この設定は、整数のピクセル数で表されます。たとえば、親ムービークリップを 50% に縮小して、_y プロパティを設定すると、ムービークリップ内のオブジェクトの移動距離は、ムービーの拡大 / 縮小率が 100% だったときの半分のピクセル数になります。

例

次の例では、実行時にムービークリップ box_mc を作成します。描画 API を使用してこのインスタンス内にボックスを描画し、ボックスの上にマウスが移動すると、ムービークリップに水平方向および垂直方向の拡大 / 縮小が適用されます。マウスがインスタンスから離れると、元の拡大 / 縮小率に戻ります。

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

関連項目

[_x \(MovieClip._x プロパティ\)](#), [_y \(MovieClip._y プロパティ\)](#), [_yscale \(MovieClip._yscale プロパティ\)](#), [_width \(MovieClip._width プロパティ\)](#)

_y (MovieClip._y プロパティ)

public _y : Number

親ムービークリップのローカル座標を基準にしてムービークリップの y 座標を設定します。ムービークリップがメインタイムラインにある場合、その座標系はステージの左上隅を (0,0) として参照します。変形を含んでいる別のムービークリップの内部にムービークリップがある場合、そのムービークリップの座標系は、それを囲む親ムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップの場合、そのムービークリップの子は、反時計回りに 90 度回転した座標系を継承します。ムービークリップの座標は、基準点の位置を参照します。

関連項目

_x (MovieClip._x プロパティ), _xscale (MovieClip._xscale プロパティ), _yscale (MovieClip._yscale プロパティ)

_ymouse (MovieClip._ymouse プロパティ)

public _ymouse : Number (読み取り専用)

マウス位置の y 座標を示します。

メモ: このプロパティは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、ステージ (_level0) 上のマウスの現在の x 座標と y 座標を、ステージ上のムービークリップ my_mc を基準とする相対位置として返します。

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = "&nbsp;\t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
    mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
    mouse_txt.htmlText += row1_str;
    mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
    mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
    mouse_txt.htmlText += "</textformat>";
};
```

関連項目

Mouse,_xmouse (MovieClip._xmouse プロパティ)

_yscale (MovieClip._yscale プロパティ)

```
public _yscale : Number
```

ムービークリップの基準点から適用するムービークリップの垂直スケール(percentage)を設定します。デフォルトの基準点は(0,0)です。

ローカル座標系を拡大 / 縮小すると、_x および _y プロパティの設定に影響します。この設定は、整数のピクセル数で表されます。たとえば、親ムービークリップを 50% に縮小した場合、_x プロパティを設定すると、ムービークリップ内のオブジェクトの移動距離は、ムービーの拡大 / 縮小率が 100% だったときの半分のピクセル数になります。

例

次の例では、実行時にムービークリップ box_mc を作成します。描画 API を使用してこのインスタンス内にボックスを描画し、ボックスの上にマウスが移動すると、ムービークリップに水平方向および垂直方向の拡大 / 縮小が適用されます。マウスがインスタンスから離れるとき、元の拡大 / 縮小率に戻ります。

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

関連項目

[_x \(MovieClip._x プロパティ\)](#), [_xscale \(MovieClip._xscale プロパティ\)](#),
[_y \(MovieClip._y プロパティ\)](#), [_height \(MovieClip._height プロパティ\)](#)

MovieClipLoader

```
Object
|
+-MovieClipLoader

public class MovieClipLoader
extends Object
```

MovieClipLoader クラスを使用すると、SWF、JPEG、GIF、および PNG の各ファイルのムービークリップへのロード時（ダウンロード時）にステータス情報を提供するリスナーコールバックを実装できます。MovieClipLoader の機能を使用するには、loadMovie() や MovieClip.loadMovie() ではなく、MovieClipLoader.loadClip() を使用して SWF ファイルをロードします。

MovieClipLoader.loadClip() メソッドを発行した後、次に列挙した順にイベントが発生します。

- ダウンロード対象ファイルの先頭バイトがディスクに書き込まれると、
MovieClipLoader.onLoadStart リスナーが呼び出されます。
- MovieClipLoader.onLoadProgress リスナーを実装している場合は、このリスナーがロードプロセスで呼び出されます。
メモ: MovieClipLoader.getProgress() はロードプロセスの任意のタイミングで呼び出すことができます。
- ダウンロード対象ファイル全体がディスクに書き込まれると、
MovieClipLoader.onLoadComplete リスナーが呼び出されます。
- ダウンロード対象ファイルの最初のフレームアクションが実行された後に、
MovieClipLoader.onLoadInit リスナーが呼び出されます。

MovieClipLoader.onLoadInit が呼び出されると、プロパティを設定したりメソッドを使用するなどの方法で、ロード済みのムービーを操作することができます。

ファイルが完全にロードされなかった場合は、MovieClipLoader.onLoadError リスナーが呼び出されます。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
<code>onLoadComplete = function (listenerObject, [target_mc]) {}</code>	<code>MovieClipLoader.loadClip()</code> でロードされたファイルが完全にダウンロードされたときに呼び出されます。
<code>onLoadError = function(target_mc, errorCode) {}</code>	<code>MovieClipLoader.loadClip()</code> でロードされたファイルがロードに失敗したときに呼び出されます。
<code>onLoadInit = function ([target_mc]) {}</code>	ロード対象の先頭フレーム上のアクションが実行されたときに呼び出されます。
<code>onLoadProgress = function ([target_mc], loadedBytes, totalBytes) {}</code>	ロードプロセス中(つまり <code>MovieClipLoader.onLoadStart</code> から <code>MovieClipLoader.onLoadComplete</code> までの間)、ロード対象のコンテンツがディスクに書き込まれるたびに呼び出されます。
<code>onLoadStart = function ([target_mc]) {}</code>	<code>MovieClipLoader.loadClip()</code> の呼び出しでファイルのダウンロードが正常に開始されたときに呼び出されます。

コンストラクター一覧

シグネチャ	説明
<code>MovieClipLoader()</code>	SWF、JPEG、GIF、または PNG の各ファイルのダウンロード時、イベントに応答するリスナーを実装するための <code>MovieClipLoader</code> オブジェクトを生成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>addListener(listener: Object) : Boolean</code>	MovieClipLoader イベントハンドラが呼び出されたときに通知を受けるオブジェクトを登録します。
	<code>getProgress(target: Object) : Object</code>	MovieClipLoader.loadClip() でロードされているファイルについて、ロード済みのバイト数および総バイト数を返します。ムービーが圧縮されている場合、getProgress メソッドが、圧縮された状態のバイト数を返します。
	<code>loadClip(url:String, target:Object) : Boolean</code>	元のムービーの再生中に、SWF ファイルまたは JPEG ファイルを Flash Player のムービークリップ内にロードします。
	<code>removeListener (listener: Object) : Boolean</code>	MovieClipLoader イベントハンドラが呼び出されたときの通知を待機するリスナーを削除します。
	<code>unloadClip(target: Object) : Boolean</code>	MovieClipLoader.loadClip() を使ってロードしたムービークリップを削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (MovieClipLoader.addListener メソッド)

public addListener(listener: Object) : Boolean

MovieClipLoader イベントハンドラが呼び出されたときに通知を受けるオブジェクトを登録します。

パラメータ

`listener: Object` - MovieClipLoader イベントハンドラからのコールバック通知を待機するオブジェクト。

戻り値

`Boolean` - ブール値。リスナーが正常に確立された場合は `true`、それ以外の場合は `false` を返します。

例

次の例では、image_mc というムービークリップにイメージをロードします。ムービークリップインスタンスは回転してステージ上に配置され、ステージおよびムービークリップの周囲に線が描かれます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = Stage.width/2-target_mc._width/2;
    target_mc._y = Stage.height/2-target_mc._width/2;
    var w:Number = target_mc._width;
    var h:Number = target_mc._height;
    target_mc.lineStyle(4, 0x000000);
    target_mc.moveTo(0, 0);
    target_mc.lineTo(w, 0);
    target_mc.lineTo(w, h);
    target_mc.lineTo(0, h);
    target_mc.lineTo(0, 0);
    target_mc._rotation = 3;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mcListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

関連項目

[onLoadComplete \(MovieClipLoader.onLoadComplete イベントリスナー\)](#), [onLoadError \(MovieClipLoader.onLoadError イベントリスナー\)](#), [onLoadInit \(MovieClipLoader.onLoadInit イベントリスナー\)](#), [onLoadProgress \(MovieClipLoader.onLoadProgress イベントリスナー\)](#), [onLoadStart \(MovieClipLoader.onLoadStart イベントリスナー\)](#), [removeListener \(MovieClipLoader.removeListener メソッド\)](#)

getProgress (MovieClipLoader.getProgress メソッド)

public getProgress(target:[Object](#)) : [Object](#)

MovieClipLoader.loadClip() でロードされているファイルについて、ロード済みのバイト数および総バイト数を返します。ムービーが圧縮されている場合、getProgress メソッドが、圧縮された状態のバイト数を返します。getProgress メソッドを使用すると、

MovieClipLoader.onLoadProgress リスナー関数を作成または追加しなくても、こういった情報を明示的に要求できます。

パラメータ

target:[Object](#) - MovieClipLoader.loadClip() を使用してロードされる SWF、JPEG、GIF、または PNG の各ファイル。

戻り値

[Object](#) - bytesLoaded および bytesTotal の 2 つの整数プロパティを保持するオブジェクト。

例

次の例では、getProgress メソッドの使用方法を示します。通常は、このメソッドを使用するのではなく、リスナーオブジェクトを作成して onLoadProgress イベントを待機します。このメソッドに関しては、getProgress の最初の同期呼び出しにより、コンテナの bytesLoaded と bytesTotal を返すことができますが、外部で要求されたオブジェクトの値は対象外であることに注意してください。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getNextHighestDepth());
var image:MovieClip = container.createEmptyMovieClip("image",
    container.getNextHighestDepth());

var mcLoader:MovieClipLoader = new MovieClipLoader();
var listener:Object = new Object();
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " +
        bytesTotal);
}
mcLoader.addListener(listener);
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", image);

var interval:Object = new Object();
interval.id = setInterval(checkProgress, 100, mcLoader, image, interval);

function checkProgress(mcLoader:MovieClipLoader, image:MovieClip,
    interval:Object):Void {
    trace(">> checking progress now with : " + interval.id);
    var progress:Object = mcLoader.getProgress(image);
    trace("bytesLoaded: " + progress.bytesLoaded + " bytesTotal: " +
        progress.bytesTotal);
    if(progress.bytesLoaded == progress.bytesTotal) {
        clearInterval(interval.id);
    }
}
```

関連項目

[loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadProgress \(MovieClipLoader.onLoadProgress イベントリスナー\)](#)

loadClip (MovieClipLoader.loadClip メソッド)

```
public loadClip(url:String, target:Object) : Boolean
```

元のムービーの再生中に、SWF ファイルまたは JPEG ファイルを Flash Player のムービークリップ内にロードします。このメソッドを使用すると、複数の SWF ファイルを同時に表示し、別の HTML ドキュメントをロードせずに SWF ファイルを切り替えることができます。

loadClip() メソッドを loadMovie() や MovieClip.loadMovie() の代わりに使用することには有利な点がいくつかあります。次のハンドラは、リスナーオブジェクトの使用によって実装されます。リスナーをアクティプにするには、MovieClipLoader.addListener(listenerObject) を使用して、MovieClipLoader クラスでリスナーを登録します。

- ロードが開始されたときに MovieClipLoader.onLoadStart ハンドラが呼び出される。
- クリップをロードできなかった場合に MovieClipLoader.onLoadError ハンドラが呼び出される。
- ロードプロセスの進行時にリアルタイムに MovieClipLoader.onLoadProgress ハンドラが呼び出される。
- ファイルのダウンロードが終了し、ロードされたムービークリップのメソッドとプロパティが使用可能になる前に、MovieClipLoader.onLoadComplete ハンドラが呼び出される。このハンドラは、onLoadInit ハンドラの前に呼び出されます。
- クリップの先頭フレームのアクションが実行された後に MovieClipLoader.onLoadInit ハンドラが呼び出される。そのため、ロード済みのクリップに対して各種の操作を開始できます。このハンドラは、onLoadComplete ハンドラの後に呼び出されます。ほとんどの場合、onLoadInit ハンドラを使用します。

ムービークリップ内にロードした SWF ファイルまたはイメージは、そのムービークリップの位置、回転、および拡大 / 縮小の各プロパティを継承します。ムービークリップのターゲットパスを使用して、ロードしたムービーをターゲットとして設定できます。

loadClip() メソッドは、単一のムービークリップまたはレベルに対して 1つまたは複数のファイルをロードするときに使用できます。MovieClipLoader リスナーオブジェクトは、ロード中のターゲットムービークリップのインスタンスにパラメータとして渡されます。代わりに、ロードするファイルごとに異なる MovieClipLoader オブジェクトを作成することもできます。

このメソッドを使用してロードされたムービーまたはイメージを削除したり、進行中のロード処理をキャンセルするには、MovieClipLoader.unloadClip() を使用します。

MovieClipLoader.getProgress() および MovieClipLoaderListener.onLoadProgress は、ファイルがローカルである場合、オーサリングプレーヤーの実際の bytesLoaded および bytesTotal の値を報告しません。オーサリング環境でプロファイル機能を使用すると、MovieClipLoader.getProgress() および MovieClipLoaderListener.onLoadProgress は、プロファイルで提供される減少した帯域幅レートではなく、実際のダウンロードレートでダウンロードを報告します。

パラメータ

url:String - ロードする SWF ファイルまたは JPEG ファイルの絶対 URL または相対 URL。相対パスは、レベル 0 の SWF ファイルが埋め込まれた HTML ファイルを基準にする必要があります。絶対 URL の場合は http:// や file:/// などのプロトコル参照を含めて指定します。ファイル名には、ドライブ指定を含めることはできません。

target:Object - ムービークリップのターゲットパス、またはムービーのロード先となる、Flash Player のレベルを指定する整数。ターゲットムービークリップは、ロードした SWF ファイルまたはイメージに置き換えられます。

戻り値

Boolean - ブール値。URL リクエストが正常に送信された場合は true、それ以外の場合は false を返します。

例

次の例では、onLoadInit イベント用のハンドラを作成し要求を行うことで、MovieClipLoader.loadClip メソッドの使用方法を示します。

次のコードを、タイムラインのフレームアクションに直接配置するか、MovieClip を拡張するクラスにペーストする必要があります。

onLoadInit イベントのハンドラメソッドを作成します。

```
public function onLoadInit(mc:MovieClip):Void {
    trace("onLoadInit: " + mc);
}
```

空のムービークリップを作成し、MovieClipLoader を使用してイメージをそのムービークリップにロードします。

```
var container:MovieClip = createEmptyMovieClip("container",
    getNextHighestDepth());
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(this);
mcLoader.loadClip("YourImage.jpg", container);

function onLoadInit(mc:MovieClip) {
    trace("onLoadInit: " + mc);
}
```

関連項目

[onLoadInit \(MovieClipLoader.onLoadInit イベントリスナー \)](#)

MovieClipLoader コンストラクタ

```
public MovieClipLoader()
```

SWF、JPEG、GIF、または PNG の各ファイルのダウンロード時、イベントに応答するリスナーを実装するための MovieClipLoader オブジェクトを生成します。

例

`MovieClipLoader.loadClip()` を参照してください。

関連項目

`addListener (MovieClipLoader.addListener メソッド)`,
`loadClip (MovieClipLoader.loadClip メソッド)`

onLoadComplete (MovieClipLoader.onLoadComplete イベントリスナー)

```
onLoadComplete = function(listenerObject, [target_mc]) {}
```

`MovieClipLoader.loadClip()` でロードされたファイルが完全にダウンロードされたときに呼び出されます。`target_mc` の値は、この呼び出しが実行されるムービークリップを識別します。これは、複数のファイルが同じリスナーセットを使用してロードされる場合に有用です。

このパラメータは、Flash によってコードに渡されますが、リスナー関数にすべてのパラメータを実装する必要はありません。

MovieClipLoader クラスで `onLoadComplete` イベントおよび `onLoadInit` イベントを使用する際には、SWF ファイルにおけるこれらの動作の違いについて理解しておくことが重要です。

`onLoadComplete` イベントは、SWF ファイルまたは JPEG ファイルがロードされた後、アプリケーションが初期化される前に呼び出されます。この時点で、ロードされたムービークリップのメソッドやプロパティにアクセスすることはできません。そのため、関数の呼び出しや、特定フレームへの移動などを行うことはできません。ほとんどの場合、代わりに `onLoadInit` イベントを使用することをお勧めします。このイベントは、コンテンツがロードされて完全に初期化された後で呼び出されます。

パラメータ

`listenerObject` - `MovieClipLoader.addListener()` を使用して追加されるリスナーオブジェクト。

`target_mc` : (オプション) - `MovieClipLoader.loadClip()` メソッドでロードされるムービークリップ。このパラメータはオプションです。

例

次の例では、image_mc というムービークリップインスタンスにイメージをロードします。onLoadInit および onLoadComplete イベントを使用して、イメージのロードにどのくらい時間がかかるかを調べます。その情報は、動的に作成される timer_txt というテキストフィールドに表示されます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
    target_mc._height, target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
112x112/box_studio_112x112.jpg", image_mc);
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadStart \(MovieClipLoader.onLoadStart イベントリスナー\)](#), [onLoadError \(MovieClipLoader.onLoadError イベントリスナー\)](#), [onLoadInit \(MovieClipLoader.onLoadInit イベントリスナー\)](#)

onLoadError (MovieClipLoader.onLoadError イベントリスナー)

onLoadError = function(target_mc, errorCode) {}
MovieClipLoader.loadClip() でロードされたファイルがロードに失敗したときに呼び出されます。このリスナーが呼び出される理由はさまざまです。たとえば、サーバーがダウンした場合、ファイルが見つからなかったり、セキュリティ侵害が発生します。

MovieClipLoader.addListener() を使用して追加するリスナーオブジェクトでこのリスナーを呼び出します。

target_mc の値は、この呼び出し対象のムービークリップを識別します。このパラメータは、複数のファイルが同じリスナーセットでロードされる場合に有用です。

`errorCode` パラメータについては、`MovieClipLoader.onLoadStart` または `MovieClipLoader.onLoadComplete` のいずれのリスナーも呼び出されなかった場合、ストリング "URLNotFound" が返されます。たとえば、サーバーがダウンしている場合やファイルが見つからなかった場合などです。`MovieClipLoader.onLoadStart` だけが呼び出され、`MovieClipLoader.onLoadComplete` は呼び出されなかった場合、ストリング "LoadNeverCompleted" が返されます。たとえば、サーバーに大きな負荷がかかったことによりダウンロードが中断された場合や、サーバーがクラッシュした場合などです。

パラメータ

`target_mc`:- `MovieClipLoader.loadClip()` メソッドでロードされるムービークリップ。

`errorCode`:- エラーの理由を示すストリング。

例

次の例では、イメージのロードが失敗すると [出力] パネルに情報を表示します。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
    trace("ERROR!");
    switch (errorCode) {
        case 'URLNotFound' :
            trace("\t Unable to connect to URL: "+target_mc._url);
            break;
        case 'LoadNeverCompleted' :
            trace("\t Unable to complete download: "+target_mc);
            break;
    }
};
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("success");
    trace(image_mc.getProgress(target_mc).bytesTotal+" bytes loaded");
};
var image_mc:MovieClipLoader = new MovieClipLoader();
image_mc.addListener(mcListener);
image_mc.loadClip("http://www.fakedomain.com/images/bad_hair_day.jpg",
    image_mc);
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadStart \(MovieClipLoader.onLoadStart イベントリスナー\)](#), [onLoadComplete \(MovieClipLoader.onLoadComplete イベントリスナー\)](#)

onLoadInit (MovieClipLoader.onLoadInit イベントリスナー)

```
onLoadInit = function([target_mc]) {}
```

ロード対象の先頭フレーム上のアクションが実行されたときに呼び出されます。このリスナーが呼び出されると、プロパティを設定したりメソッドを使用するなどの方法で、ロード済みのムービーを操作することができます。MovieClipLoader.addListener() を使用して追加するリスナーオブジェクトでこのリスナーを呼び出します。

target_mc の値は、この呼び出し対象のムービークリップを識別します。このパラメータは、複数のファイルが同じリスナーセットでロードされる場合に有用です。

パラメータ target_mc: MovieClip(オプション)-MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

パラメータ

target_mc:(オプション)-MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

例

次の例では、image_mc というムービークリップインスタンスにイメージをロードします。

onLoadInit および onLoadComplete イベントを使用して、イメージのロードにどのくらい時間がかかるかを調べます。その情報は、timer_txt というテキストフィールドに表示されます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
        target_mc._height,
        target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

次の例では、実行時に作成済みのムービークリップにムービーがロードされたかどうかを確認します。

```
this.createEmptyMovieClip("tester_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("movie loaded");
}
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.yourserver.com/your_movie.swf", tester_mc);
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadStart \(MovieClipLoader.onLoadStart イベントリスナー\)](#)

onLoadProgress (MovieClipLoader.onLoadProgress イベントリスナー)

onLoadProgress = function([target_mc], loadedBytes, totalBytes) {}

ロードプロセス中(つまり MovieClipLoader.onLoadStart から

MovieClipLoader.onLoadCompleteまでの間)、ロード対象のコンテンツがディスクに書き込まれるたびに呼び出されます。MovieClipLoader.addListener() を使用して追加するリスナーオブジェクトでこのリスナーを呼び出します。このメソッドの loadedBytes パラメータおよび totalBytes パラメータを使用することにより、ダウンロードの進捗情報を表示できます。

target_mc の値は、この呼び出し対象のムービークリップを識別します。これは、複数のファイルが同じリスナーセットでロードされる場合に有用です。

メモ: onLoadProgress をプレビューモードで使用してハードディスク上にあるローカルファイルをロードしようとしても、プレビューモードではローカルファイル全体がロードされるため、正常に機能しません。

パラメータ target_mc: MovieClip [optional] - MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

loadedBytes: Number - リスナーが呼び出された時点で既にロードされていたバイト数。

totalBytes: Number - ロード対象ファイルの総バイト数。

パラメータ

target_mc: (オプション) - MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

loadedBytes: - リスナーが呼び出された時点で既にロードされていたバイト数。

totalBytes: - ロード対象ファイルの総バイト数。

例

次の例では、新しいムービークリップ、新しいMovieClipLoader、および匿名イベントリスナーを作成します。ロードの進行状況を定期的に出力し、ロードが完了して ActionScript でアセットを使用できるようになると最終的な通知を発行します。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getNextHighestDepth());
var mcLoader:MovieClipLoader = new MovieClipLoader();
var listener:Object = new Object();
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " +
        bytesTotal);
}
listener.onLoadInit = function(target:MovieClip):Void {
    trace(target + ".onLoadInit");
}
mcLoader.addListener(listener);
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", container);
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [getProgress \(MovieClipLoader.getProgress メソッド\)](#)

onLoadStart (MovieClipLoader.onLoadStart イベントリスナー)

```
onLoadStart = function([target_mc]) {}
```

MovieClipLoader.loadClip() の呼び出しでファイルのダウンロードが正常に開始されたときに呼び出されます。MovieClipLoader.addListener() を使用して追加するリスナーオブジェクトでこのリスナーを呼び出します。

target_mc の値は、この呼び出し対象のムービークリップを識別します。このパラメータは、複数のファイルが同じリスナーセットでロードされる場合に有用です。

パラメータ target_mc: MovieClip [optional] - MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

パラメータ

target_mc:(オプション)-MovieClipLoader.loadClip() メソッドでロードされるムービークリップ。

例

次の例では、image_mc というムービークリップインスタンスにイメージをロードします。onLoadInit および onLoadComplete イベントを使用して、イメージのロードにどのくらい時間がかかるかを調べます。その情報は、timer_txt というテキストフィールドに表示されます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
        target_mc._height,
        target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#), [loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadError \(MovieClipLoader.onLoadError イベントリスナー\)](#), [onLoadInit \(MovieClipLoader.onLoadInit イベントリスナー\)](#), [onLoadComplete \(MovieClipLoader.onLoadComplete イベントリスナー\)](#)

removeListener (MovieClipLoader.removeListener メソッド)

public removeListener(*listener:Object*) : Boolean

MovieClipLoader イベントハンドラが呼び出されたときの通知を待機するリスナーを削除します。以降のロード画面は表示されません。

パラメータ

listener:Object - MovieClipLoader.addListener() を使用して追加されるリスナーオブジェクト。

戻り値

[Boolean](#) -

例

次の例では、ムービークリップにイメージをロードし、ユーザーが start_button および stop_button という 2 つのボタンを使用してロードを開始および停止できるようにします。ロードを開始または停止すると、情報が [出力] パネルに表示されます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadStart = function(target_mc:MovieClip) {
    trace("\t onLoadStart");
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
    trace("\t onLoadComplete");
};
mcListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
    trace("\t onLoadError: "+errorCode);
};
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("\t onLoadInit");
    start_button.enabled = true;
    stop_button.enabled = false;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
//
start_button.clickHandler = function() {
    trace("Starting...");
    start_button.enabled = false;
    stop_button.enabled = true;
    //
    image_mcl.addListener(mcListener);
    image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
        image_mc);
};
stop_button.clickHandler = function() {
    trace("Stopping...");
    start_button.enabled = true;
    stop_button.enabled = false;
    //
    image_mcl.removeListener(mcListener);
};
stop_button.enabled = false;
```

関連項目

[addListener \(MovieClipLoader.addListener メソッド\)](#)

unloadClip (MovieClipLoader.unloadClip メソッド)

public unloadClip(target:[Object](#)) : Boolean

MovieClipLoader.loadClip() を使ってロードしたムービークリップを削除します。このメソッドをムービーのロード中に発行すると、MovieClipLoader.onLoadError が呼び出されます。

パラメータ

target:[Object](#) - my_mc1.loadClip() に対応する呼び出しに渡されるストリングまたは整数。

戻り値

Boolean -

例

次の例では、image_mc というムービークリップにイメージをロードします。ムービークリップをクリックすると、そのムービークリップが削除され、情報が[出力]パネルに表示されます。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = 100;
    target_mc._y = 100;
    target_mc.onRelease = function() {
        trace("Unloading clip...");
        trace("\t name: "+target_mc._name);
        trace("\t url: "+target_mc._url);
        image_mc1.unloadClip(target_mc);
    };
}
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

関連項目

[loadClip \(MovieClipLoader.loadClip メソッド\)](#), [onLoadError \(MovieClipLoader.onLoadError イベントリスナー\)](#)

Number

```
Object
 |
 +- Number
```

```
public class Number
extends Object
```

Number クラスは、Number データ型の単純なラッパーオブジェクトです。Number クラスに関連するメソッドとプロパティを使用してプリミティブな数値を処理することができます。このクラスは、JavaScript の Number クラスと同じです。

Number クラスのプロパティは静的であるため、プロパティを使用するためのオブジェクトは不要で、コンストラクタを使用する必要はありません。

次の例では、Number クラスの `toString()` メソッドを呼び出します。このメソッドはストリング 1234 を返します。

```
var myNumber:Number = new Number(1234);
myNumber.toString();
```

次の例では、コンストラクタを使用せずに、`MIN_VALUE` プロパティの値を宣言された変数に割り当てます。

```
var smallest:Number = Number.MIN_VALUE;
```

プロパティ一覧

オプション	プロパティ	説明
static	<code>MAX_VALUE: Number</code>	表現可能な最大の数値 (倍精度 IEEE-754)。
static	<code>MIN_VALUE: Number</code>	表現可能な最小の数値 (倍精度 IEEE-754)。
static	<code>NaN: Number</code>	非数 (NaN) を表す IEEE-754 の値。
static	<code>NEGATIVE_INFINITY: Number</code>	負の無限大を表す IEEE-754 値を指定します。
static	<code>POSITIVE_INFINITY: Number</code>	正の無限大を表す IEEE-754 値を指定します。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve
プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>Number(num:Object)</code>	新しい Number オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>toString(radix:Number) : String</code>	指定された Number オブジェクト (<i>myNumber</i>) のストリング表現を返します。
	<code>valueOf() : Number</code>	指定された Number オブジェクトのプリミティブな値のタイプを返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

MAX_VALUE (Number.MAX_VALUE プロパティ)

public static MAX_VALUE : Number

表現可能な最大の数値 (倍精度 IEEE-754)。この数値は、約 1.79e+308 です。

例

次の ActionScript は、表現可能な最大の数値および最小の数値を [出力] パネルに表示します。

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);  
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

このコードは、次の値を表示します。

```
Number.MIN_VALUE = 4.94065645841247e-324  
Number.MAX_VALUE = 1.79769313486232e+308
```

MIN_VALUE (Number.MIN_VALUE プロパティ)

```
public static MIN_VALUE : Number
```

表現可能な最小の数値(倍精度 IEEE-754)。この数値は、約 5e-324 です。

例

次の ActionScript は、表現可能な最大の数値および最小の数値を [出力] パネルおよびログファイルに表示します。

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

このコードは、次の値を表示します。

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

NaN (Number.NaN プロパティ)

```
public static NaN : Number
```

非数 (NaN) を表す IEEE-754 の値。

関連項目

[isNaN 関数](#)

NEGATIVE_INFINITY

(Number.NEGATIVE_INFINITY プロパティ)

```
public static NEGATIVE_INFINITY : Number
```

負の無限大を表す IEEE-754 値を指定します。このプロパティの値は、-Infinity 定数の値と同じです。

負の無限大は、数学演算または関数が表現できる下限を超える負の値を返すときに返される特別な数値です。

例

次の例では、以下の値の除算結果を比較します。

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
    trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

Number コンストラクタ

```
public Number(num:Object)
```

新しい Number オブジェクトを作成します。new Number コンストラクタは、主にプレースホルダとして使用します。Number オブジェクトは、パラメータをプリミティブ値に変換する Number() 関数とは異なります。

パラメータ

`num:Object` - 作成される Number オブジェクトの数値、または数値に変換される値。`value` が指定されなかった場合のデフォルト値は 0 です。

例

次のコードは、新しい Number オブジェクトを作成します。

```
var n1:Number = new Number(3.4);
var n2:Number = new Number(-10);
```

関連項目

`toString (Number.toString メソッド), valueOf (Number.valueOf メソッド)`

POSITIVE_INFINITY (Number.POSITIVE_INFINITY プロパティ)

```
public static POSITIVE_INFINITY : Number
```

正の無限大を表す IEEE-754 値を指定します。このプロパティの値は、`Infinity` 定数の値と同じです。

正の無限大は、数学演算または関数が表現できる上限を超える正の値を返すときに返される特別な数値です。

例

次の例では、以下の値の除算結果を比較します。

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
    trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

toString (Number.toString メソッド)

```
public toString(radix:Number) : String
```

指定された Number オブジェクト (*myNumber*) のストリング表現を返します。

パラメータ

radix: Number - 数値からストリングへの変換に使用する基数 (2 ~ 36) を指定します。 radix パラメータを指定しない場合、デフォルト値は 10 です。

戻り値

String - ストリング。

例

次の例では、 radix パラメータに 2 および 8 を使用し、数値 9 に対応する表現を含むストリングを返します。

```
var myNumber:Number = new Number(9);
trace(myNumber.toString(2)); // output: 1001
trace(myNumber.toString(8)); // output: 11
```

次の例では、結果が 16 進数値になります。

```
var r:Number = new Number(250);
var g:Number = new Number(128);
var b:Number = new Number(114);
var rgb:String = "0x"+ r.toString(16)+g.toString(16)+b.toString(16);
trace(rgb);
// output: rgb:0xFA8072 (Hexadecimal equivalent of the color 'salmon')
```

valueOf (Number.valueOf メソッド)

```
public valueOf() : Number
```

指定された Number オブジェクトのプリミティブな値のタイプを返します。

戻り値

Number - ストリング。

例

次の例では、 numSocks オブジェクトのプリミティブな値が結果として返されます。

```
var numSocks = new Number(2);
trace(numSocks.valueOf()); // output: 2
```

Object

Object

```
public class Object
```

Object クラスは、ActionScript クラス階層のルートにあります。このクラスは、JavaScript Object クラスで提供される機能の小さいサブセットから構成されます。

プロパティ一覧

オプション	プロパティ	説明
	<code>constructor:Object</code>	特定のオブジェクトインスタンスのコンストラクタ関数を参照します。
	<code>__proto__:Object</code>	オブジェクトの作成に使用された、クラスの prototype プロパティ (ActionScript 2.0) またはコンストラクタ関数 (ActionScript 1.0) を参照します。
static	<code>prototype:Object</code>	クラスまたは関数オブジェクトのスーパークラスを参照します。
	<code>__resolve:Object</code>	ActionScript コードが未定義のメソッドまたはプロパティを参照する場合に自動的に呼び出されるユーザー定義関数への参照です。

コンストラクター一覧

シグネチャ	説明
<code>Object()</code>	Object オブジェクトを作成し、そのオブジェクトのコンストラクタメソッドへの参照をオブジェクトの constructor プロパティに格納します。

メソッド一覧

オプション	シグネチャ	説明
	<code>addProperty (name:String, getter:Function, setter:Function) : Boolean</code>	getter/setter プロパティを作成します。
	<code>hasOwnProperty(name: String) : Boolean</code>	オブジェクトに指定されたプロパティが定義されているかどうかを示します。

オプション	シグネチャ	説明
	<code>isPropertyEnumerable (name:String) : Boolean</code>	指定されたプロパティが存在し列挙できるかどうかを示します。
	<code>isPrototypeOf (theClass:Object) : Boolean</code>	Object クラスのインスタンスが、パラメータとして指定されたオブジェクトのプロトタイプチェーン内にあるかどうかを示します。
static	<code>registerClass (name:String, theClass:Function) : Boolean</code>	ムービークリップシンボルと ActionScript オブジェクトクラスを関連付けます。
	<code>toString() : String</code>	指定されたオブジェクトをストリングに変換し、返します。
	<code>unwatch (name:String) : Boolean</code>	<code>Object.watch()</code> で作成した監視ポイントを削除します。
	<code>valueOf() : Object</code>	指定されたオブジェクトのプリミティブな値を返します。
	<code>watch(name:String, callback:Function, [userData:Object]) : Boolean</code>	ActionScript オブジェクトの指定されたプロパティの変更に応じて呼び出されるイベントハンドラを登録します。

addProperty (Object.addProperty メソッド)

`public addProperty(name:String, getter:Function, setter:Function) : Boolean`

getter/setter プロパティを作成します。Flash は getter/setter プロパティを読み込むと、get 関数を呼び出します。この関数の戻り値は name の値になります。Flash は getter/setter プロパティを書き込むと、set 関数を呼び出し、それに新しい値をパラメータとして渡します。指定された名前のプロパティが既に存在する場合は、新しいプロパティによって上書きされます。

get 関数はパラメータがない関数です。戻り値のタイプは不特定です。戻り値のタイプは呼び出しごとに変わることがあります。戻り値は、プロパティの現在の値として扱われます。

set 関数のパラメータは 1 つであり、プロパティの新しい値です。たとえば、プロパティ x がステートメント `x = 1` によって割り当てられた場合、set 関数には Number 型のパラメータ 1 が渡されます。set 関数の戻り値は無視されます。

getter/setter プロパティはプロトタイプオブジェクトに追加できます。getter/setter プロパティをプロトタイプオブジェクトに追加すると、プロトタイプオブジェクトを継承するすべてのオブジェクトインスタンスは getter/setter プロパティを継承します。つまり、1つのプロトタイプオブジェクトにgetter/setter プロパティを追加するだけで、クラスのすべてのインスタンスに同じプロパティを追加できます（プロトタイプオブジェクトにメソッドを追加するのと似ています）。継承先のプロトタイプオブジェクトの getter/setter プロパティに対して呼び出される get/set 関数には、このプロトタイプオブジェクトへの参照ではなく、継承元のオブジェクトへの参照が渡されます。

正常に呼び出されなかった場合は、Object.addProperty() が失敗し、エラーが発生することがあります。次の表は、発生する可能性があるエラーの一覧です。

エラー状態	結果
name が有効なプロパティ名ではありません。 たとえば、空のストリングです。	false が返され、プロパティは追加されません。
getter は有効な関数オブジェクトではありません。	false が返され、プロパティは追加されません。
setter は有効な関数オブジェクトではありません。	false が返され、プロパティは追加されません。

パラメータ

`name: String` - 作成対象のオブジェクトプロパティの名前を示すストリング。

`getter: Function` - プロパティの値を取得するために呼び出される関数。このパラメータは Function オブジェクトです。

`setter: Function` - プロパティの値を設定するために呼び出される関数。このパラメータは Function オブジェクトです。このパラメータに値 `null` を渡すと、プロパティは読み取り専用になります。

戻り値

`Boolean` - ブール値。プロパティが正常に作成された場合は `true` を返します。それ以外の場合は `false` を返します。

例

次の例では、オブジェクトに2つの内部メソッド `setQuantity()` および `getQuantity()` があります。`bookcount` というプロパティの値を設定または取得するときに、これらのメソッドが呼び出されます。もう1つの `getTitle()` という内部メソッドは、`bookname` プロパティに設定されている読み取り専用の値を返します。スクリプトが `myBook.bookcount` の値を取得すると、ActionScript インタプリタは `myBook.getQuantity()` を自動的に呼び出します。スクリプトで `myBook.bookcount` の値を修正すると、インタプリタは `myObject.setQuantity()` を呼び出します。`bookname` プロパティでは `set` 関数を指定していません。したがって、`bookname` プロパティ値を変更する試みは無視されます。

```
function Book() {
    this.setQuantity = function(numBooks:Number):Void {
        this.books = numBooks;
    };
    this.getQuantity = function():Number {
        return this.books;
    };
    this.getTitle = function():String {
        return "Catcher in the Rye";
    };
    this.addProperty("bookcount", this.getQuantity, this.setQuantity);
    this.addProperty("bookname", this.getTitle, null);
}
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
// output: You ordered 5 copies of Catcher in the Rye
```

前の例は有効ですが、`bookcount` および `bookname` プロパティが `Book` オブジェクトの各インスタンスに追加されます。オブジェクトの各インスタンスで2つのプロパティが必須となります。クラスに `bookcount` や `bookname` のようなプロパティが多数ある場合は、多大なメモリを消費します。代わりの方法として、`bookcount` および `bookname` プロパティを1つの場所にだけ存在させるために、プロパティを `Book.prototype` に追加します。このようにしても、結果は `bookcount` プロパティと `bookname` プロパティをすべてのインスタンスに直接追加したシンタックス例のコードと同じです。`Book` インスタンスのいずれかのプロパティにアクセスした場合に、プロパティが存在しなければ、プロトタイプチェーンを上にたどることによって `Book.prototype` で定義されたバージョンが見つかります。次の例では、プロパティを `Book.prototype` に追加する方法を示します。

```
function Book() {}
Book.prototype.setQuantity = function(numBooks:Number):Void {
    this.books = numBooks;
};
Book.prototype.getQuantity = function():Number {
    return this.books;
};
Book.prototype.getTitle = function():String {
    return "Catcher in the Rye";
```

```
};

Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
    Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

次の例では、ActionScript 2.0 で利用可能な暗黙的な **getter** および **setter** 関数を使用する方法を示しています。Book 関数を定義して Book.prototype を編集する代わりに、"Book.as" という名前の外部ファイルに Book クラスを定義します。次のコードは、"Book.as" という名前の別の外部ファイルに配置する必要があります。このファイルは、Flash アプリケーションのクラスパスに含まれ、このクラス定義のみを格納します。

```
class Book {
    var books:Number;
    function set bookcount(numBooks:Number):Void {
        this.books = numBooks;
    }
    function get bookcount():Number {
        return this.books;
    }
    function get bookname():String {
        return "Catcher in the Rye";
    }
}
```

次のコードは、FLA ファイルに配置することができ、前の例の場合と同じ機能を果します。

```
var myBook:Book = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

関連項目

[get ステートメント](#), [set ステートメント](#)

constructor (Object.constructor プロパティ)

```
public constructor : Object
```

特定のオブジェクトインスタンスのコンストラクタ関数を参照します。constructor プロパティは、Object クラスのコンストラクタを使用して作成されると、すべてのオブジェクトに自動的に割り当てられます。

例

次の例は、myObject オブジェクトのコンストラクタ関数への参照です。

```
var my_str:String = new String("sven");
trace(my_str.constructor == String); //output: true
```

`instanceof` オペレータを使用すると、オブジェクトが指定したクラスに属するかどうかを調べることができます。

```
var my_str:String = new String("sven");
trace(my_str instanceof String); //output: true
```

しかし、次の例では `Object.constructor` プロパティがプリミティブデータ型(次のようなストリングリテラルなど)をラッパーオブジェクトに変換します。次の例のように、`instanceof` オペレータは変換を行いません。

```
var my_str:String = "sven";
trace(my_str.constructor == String); //output: true
trace(my_str instanceof String); //output: false
```

関連項目

[instanceof 演算子](#)

hasOwnProperty (Object.hasOwnProperty メソッド)

```
public hasOwnProperty(name:String) : Boolean
```

オブジェクトに指定されたプロパティが定義されているかどうかを示します。ターゲットオブジェクトに `name` パラメータで指定されたストリングに一致するプロパティがある場合は `true` を返します。それ以外の場合は `false` を返します。このメソッドは、オブジェクトのプロトタイプチェーンをチェックせず、オブジェクト自身にプロパティが存在する場合にのみ `true` を返します。

パラメータ

`name:String` -

戻り値

`Boolean` - ブール値。ターゲットオブジェクトに `name` パラメータで指定されたプロパティがある場合は `true`、それ以外の場合は `false` を返します。

isPrototypeOf

(Object.isPrototypeOf メソッド)

```
public isPrototypeOf(name:String) : Boolean
```

指定されたプロパティが存在し列挙できるかどうかを示します。`true` の場合、このプロパティが存在し、`for..in` ループで列挙できます。このメソッドではターゲットオブジェクトのプロトタイプチェーンをチェックしないため、プロパティがターゲットオブジェクト上に存在している必要があります。

作成するプロパティは列挙できますが、ビルトインプロパティは通常列挙できません。

パラメータ

`name:String` -

戻り値

`Boolean` - ブール値。`name` パラメータで指定されたプロパティが列挙可能な場合は `true` となります。

例

次の例では、汎用オブジェクトを作成し、プロパティをオブジェクトに追加してから、オブジェクトが列挙可能かどうかを確認します。この例では、対比のために、`Array.length` プロパティというビルトインプロパティが列挙できないことも示します。

```
var myObj:Object = new Object();
myObj.prop1 = "hello";
trace(myObj.isPropertyEnumerable("prop1")); // Output: true

var myArray = new Array();
trace(myArray.isPropertyEnumerable("length")); // Output: false
```

関連項目

[for..in ステートメント](#)

isPrototypeOf (Object.isPrototypeOf メソッド)

`public isPrototypeOf(theClass:Object) :Boolean`

`Object` クラスのインスタンスが、パラメータとして指定されたオブジェクトのプロトタイプチェーン内にあるかどうかを示します。このメソッドは、オブジェクトが `theClass` パラメータで指定されたオブジェクトのプロトタイプチェーン内にある場合に `true` を返します。ターゲットオブジェクトが `theClass` オブジェクトのプロトタイプチェーン内にない場合はもちろん、`theClass` パラメータがオブジェクトでない場合も、`false` を返します。

パラメータ

`theClass:Object` -

戻り値

`Boolean` - ブール値。オブジェクトが `theClass` パラメータで指定されたオブジェクトのプロトタイプチェーン内にある場合は `true`、それ以外の場合は `false` を返します。

Object コンストラクタ

```
public Object()
```

Object オブジェクトを作成し、そのオブジェクトのコンストラクタメソッドへの参照をオブジェクトの constructor プロパティに格納します。

例

次の例では、myObject という名前の汎用オブジェクトを作成します。

```
var myObject:Object = new Object();
```

__proto__ (Object.__proto__ プロパティ)

```
public __proto__: Object
```

オブジェクトの作成に使用された、クラスの prototype プロパティ (ActionScript 2.0) またはコンストラクタ関数 (ActionScript 1.0) を参照します。__proto__ プロパティは、作成したすべてのオブジェクトに自動的に割り当てられます。ActionScript インタプリタは、__proto__ プロパティを使用してオブジェクトのクラスまたはコンストラクタ関数の prototype プロパティにアクセスし、オブジェクトがそのスーパークラスから継承しているプロパティとメソッドを確認します。

例

次の例では、Shape という名前のクラスと Circle という名前の Shape のスーパークラスを作成します。

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

Circle クラスを使用して、Circle の 2 つのインスタンスを作成できます。

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

次の trace ステートメントでは、両方のインスタンスの __proto__ プロパティが Circle クラスの prototype プロパティを参照していることを示します。

```
trace(Circle.prototype == oneCircle.__proto__); // Output: true
trace(Circle.prototype == twoCircle.__proto__); // Output: true
```

関連項目

[prototype \(Object.prototype プロパティ\)](#)

prototype (Object.prototype プロパティ)

```
public static prototype : Object
```

クラスまたは関数オブジェクトのスーパークラスを参照します。prototype プロパティは自動的に作成され、作成したクラスまたは関数オブジェクトに割り当てられます。このプロパティは、作成したクラスまたは関数に固有であるという点で静的です。たとえば、カスタムクラスを作成すると、prototype プロパティの値は、クラスのすべてのインスタンスで共有され、クラスプロパティとしてのみアクセスできます。カスタムクラスのインスタンスは、prototype プロパティに直接アクセスできません。アクセスするには、`__proto__` プロパティ経由で行います。

例

次の例では、Shape という名前のクラスと Circle という名前の Shape のスーパークラスを作成します。

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

Circle クラスを使用して、Circle の 2 つのインスタンスを作成できます。

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

次の trace ステートメントでは、Circle クラスの prototype プロパティがそのスーパークラス Shape をポイントしていることを示します。識別子 Shape は、Shape クラスのコンストラクタ関数を参照しています。

```
trace(Circle.prototype.constructor == Shape); // Output: true
```

次の trace ステートメントでは、prototype プロパティと `__proto__` プロパティを同時に使用して、継承階層(またはプロトタイプチェーン)を 2 レベル上に移動する方法を示します。

`Circle.prototype.__proto__` プロパティには、Shape クラスのスーパークラスへの参照が格納されています。

```
trace(Circle.prototype.__proto__ == Shape.prototype); // Output: true
```

関連項目

[__proto__ \(Object.__proto__ プロパティ \)](#)

registerClass (Object.registerClass メソッド)

```
public static registerClass(name:String, theClass:Function) : Boolean
```

ムービークリップシンボルと ActionScript オブジェクトクラスを関連付けます。シンボルが存在しない場合は、ストリング識別子とオブジェクトクラスが関連付けられます。

指定されたムービークリップシンボルのインスタンスがタイムラインで挿入された場合、そのシンボルは MovieClip クラスではなく、theClass パラメータで指定されたクラスに登録されます。

指定したムービークリップシンボルのインスタンスが MovieClip.attachMovie() または MovieClip.duplicateMovieClip() によって作成された場合は、MovieClip クラスではなく、theClass で指定したクラスに登録されます。theClass が null である場合、このメソッドは指定されたムービークリップシンボルまたはクラス識別子に関連する ActionScript クラス定義を削除します。ムービークリップシンボルの場合、ムービークリップの既存のインスタンスは変更されませんが、シンボルの新しいインスタンスはデフォルトクラス MovieClip に関連付けられます。

シンボルが既にクラスに登録されている場合は、それを新しい登録に置き換えます。

ムービークリップインスタンスがタイムラインで挿入されたか、attachMovie() または duplicateMovieClip() で作成された場合、ActionScript はそのオブジェクトを指すキーワード this を使用して対応するクラスのコンストラクタを呼び出します。コンストラクタ関数の呼び出しにはパラメータを使用しません。

このメソッドを使って、MovieClip 以外の ActionScript クラスにムービークリップを登録すると、新しいクラスのプロトタイプチェーンに MovieClip クラスを含めない限り、ムービークリップシンボルは、ビルトイン MovieClip クラスのメソッド、プロパティ、およびイベントを継承しません。次のコードでは、MovieClip クラスのプロパティを継承する theClass という新しい ActionScript クラスを作成します。

```
theClass.prototype = new MovieClip();
```

パラメータ

name:String - ストリング。ムービークリップシンボルのリンクエージ識別子、または ActionScript クラスのストリング識別子。

theClass:Function - ActionScript クラスのコンストラクタ関数への参照、またはシンボルを登録解除する null。

戻り値

Boolean - ブール値。クラスが正常に登録されると、true が返されます。それ以外は、false が返されます。

関連項目

[attachMovie \(MovieClip.attachMovie メソッド\)](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip メソッド\)](#)

_resolve(Object._resolve プロパティ)

```
public _resolve : Object
```

ActionScript コードが未定義のメソッドまたはプロパティを参照する場合に自動的に呼び出されるユーザー定義関数への参照です。ActionScript コードがオブジェクトの未定義のプロパティまたはメソッドを参照する場合、Flash Player はオブジェクトの __resolve プロパティが定義されているかどうかを判断します。__resolve が定義されていると、参照先の関数が実行され、未定義のプロパティまたはメソッドの名前が渡されます。これにより、プロパティまたはメソッドが実際に定義された場合と同じように、未定義のプロパティの値および未定義のメソッドのステートメントをプログラムによって提供できます。このプロパティは、クライアント / サーバー間で非常に透過的なやり取りが行われるようにするために有用であり、サーバーサイドメソッドを呼び出す方法として推奨されています。

例

次の例は最初の例から段階的に構築されており、__resolve プロパティの 5 とおりのシンタックスを示しています。わかりやすくするために、前のシンタックスと異なるキーステートメントは太字になっています。

シンタックス 1: 次の例では、__resolve を使用し、未定義のプロパティがそれぞれ "Hello, world!" という値を返すオブジェクトを構築しています。

```
// instantiate a new object
var myObject:Object = new Object();

// define the __resolve function
myObject.__resolve = function (name) {
    return "Hello, world!";
};

trace (myObject.property1); // output: Hello, world!
trace (myObject.property2); // output: Hello, world!
```

シンタックス 2: 次の例では、__resolve をファンクター(関数を生成する関数)として使用しています。__resolve を使用することで、未定義のメソッドの呼び出しが、myFunction という名前の汎用関数にリダイレクトされます。

```
// instantiate a new object
var myObject:Object = new Object();

// define a function for __resolve to call
myObject.myFunction = function (name) {
    trace("Method " + name + " was called");
};

// define the __resolve function
myObject.__resolve = function (name) {
    return function () { this.myFunction(name); };
};
```

```
// test __resolve using undefined method names
myObject.someMethod(); // output: Method someMethod was called
myObject.someOtherMethod(); //output: Method someOtherMethod was called

シンタックス3:次の例は、前の例を基にして構築されており、解決されたメソッドをキャッシュに保存する機能を追加しています。メソッドをキャッシュに保存すると、__resolveが各メソッドに対して呼び出されるのは1回だけです。したがって、オブジェクトメソッドの遅延構築が可能です。遅延構築は、メソッドが最初に使用されるときまで、メソッドの作成(構築)を延期する最適化技術です。
```

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};

// define the __resolve function
myObject.__resolve = function(name) {
    trace("Resolve called for "+name); // to check when __resolve is called
    // Not only call the function, but also save a reference to it
    var f:Function = function () {
        this.myFunction(name);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference
    return f;
};

// test __resolve using undefined method names
// __resolve will only be called once for each method name
myObject.someMethod(); // calls __resolve
myObject.someMethod(); // does not call __resolve because it is now defined
myObject.someOtherMethod(); // calls __resolve
myObject.someOtherMethod(); // does not call __resolve, no longer undefined
```

シンタックス4:次の例は前の例を基にして構築されており、メソッド名onStatus()をローカルで使用するために予約し、他の未定義のプロパティと同じ方法で解決されないようにしています。追加されたコードは太字になっています。

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};

// define the __resolve function
myObject.__resolve = function(name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
```

```

}

trace("Resolve called for "+name); // to check when __resolve is called
// Not only call the function, but also save a reference to it
var f:Function = function () {
    this.myFunction(name);
};

// create a new object method and assign it the reference
this[name] = f;
// return the reference
return f;
};

// test __resolve using the method name "onStatus"
trace(myObject.onStatus("hello"));
// output: undefined

シンタックス5:次の例は前の例を基にして構築されており、パラメータを受け入れるファンクターを作成しています。この例では、argumentsオブジェクトのシンタックスを拡張し、Arrayクラスのメソッドをいくつか使用しています。

// instantiate a new object
var myObject:Object = new Object();

// define a generic function for __resolve to call
myObject.myFunction = function (name) {
    arguments.shift();
    trace("Method " + name + " was called with arguments: " +
    arguments.join(',');
};

// define the __resolve function
myObject.__resolve = function (name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
    }
    var f:Function = function () {
        arguments.unshift(name);
        this.myFunction.apply(this, arguments);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference to the function
    return f;
};

// test __resolve using undefined method names with parameters
myObject.someMethod("hello");
// output: Method someMethod was called with arguments: hello

myObject.someOtherMethod("hello","world");
// output: Method someOtherMethod was called with arguments: hello,world

```

関連項目

[arguments](#), [Array](#)

toString (Object.toString メソッド)

public `toString()` : String

指定されたオブジェクトをストリングに変換し、返します。

戻り値

[String](#) - ストリング。

例

この例では、汎用オブジェクトの `toString()` に対する戻り値を示しています。

```
var myObject:Object = new Object();
trace(myObject.toString()); // output: [object Object]
```

このメソッドを上書きし、さらに意味のある値を返すことができます。次の例では、このメソッドがビルトインクラスの `Date`、`Array`、および `Number` に対して上書きされたことを示しています。

```
// Date.toString() returns the current date and time
var myDate:Date = new Date();
trace(myDate.toString()); // output: [current date and time]
```

```
// Array.toString() returns the array contents as a comma-delimited string
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two
```

```
// Number.toString() returns the number value as a string
// Because trace() won't tell us whether the value is a string or number
// we will also use typeof() to test whether toString() works.
var myNumber:Number = 5;
trace(typeof(myNumber)); // output: number
trace(myNumber.toString()); // output: 5
trace(typeof(myNumber.toString())); // output: string
```

次の例では、カスタムクラスの `toString()` を上書きする方法を示しています。最初に、"Vehicle.as" という名前のテキストファイルを作成し、そのファイルに `Vehicle` クラス定義のみを含めて、設定フォルダ内の "Classes" フォルダに格納します。

```
// contents of Vehicle.as
class Vehicle {
    var numDoors:Number;
    var color:String;
    function Vehicle(param_numDoors:Number, param_color:String) {
        this.numDoors = param_numDoors;
        this.color = param_color;
    }
}
```

```

function toString():String {
    var doors:String = "door";
    if (this.numDoors > 1) {
        doors += "s";
    }
    return ("A vehicle that is " + this.color + " and has " + this.numDoors + " "
    + doors);
}
}

// code to place into a FLA file
var myVehicle:Vehicle = new Vehicle(2, "red");
trace(myVehicle.toString());
// output: A vehicle that is red and has 2 doors

// for comparison purposes, this is a call to valueOf()
// there is no primitive value of myVehicle, so the object is returned
// giving the same output as toString().
trace(myVehicle.valueOf());
// output: A vehicle that is red and has 2 doors

```

unwatch (Object.unwatch メソッド)

`public unwatch(name:String) : Boolean`

`Object.watch()` で作成した監視ポイントを削除します。このメソッドは、監視ポイントが正常に削除されると `true` を返します。それ以外は、`false` を返します。

パラメータ

`name:String` - 監視対象から外すオブジェクトプロパティの名前を示すストリング。

戻り値

`Boolean` - ブール値。監視ポイントが正常に削除された場合は `true` を返します。それ以外の場合は `false` を返します。

例

`Object.watch()` の例を参照してください。

関連項目

[watch \(Object.watch メソッド\)](#), [addProperty \(Object.addProperty メソッド\)](#)

valueOf (Object.valueOf メソッド)

```
public valueOf() : Object
```

指定されたオブジェクトのプリミティブな値を返します。オブジェクトにプリミティブな値がない場合、オブジェクトが返されます。

戻り値

[Object](#) - 指定したオブジェクトのプリミティブ値、またはオブジェクトそのもの。

例

次の例では、汎用オブジェクト（プリミティブな値を持たない）の valueOf() の戻り値を示し、[toString\(\)](#) の戻り値と比較しています。最初に、汎用オブジェクトを作成します。次に、新しい Date オブジェクトを 2004 年 2 月 1 日午前 8 時 15 分に設定して作成します。[toString\(\)](#) メソッドは、現在の時刻を人間が理解できる形式で返します。valueOf() メソッドは、プリミティブ値をミリ秒単位で返します。次に、2 つの単純なエレメントを含む新しい Array オブジェクトを作成します。[toString\(\)](#) と valueOf() は同じ値 one,two を返します。

```
// Create a generic object
var myObject:Object = new Object();
trace(myObject.valueOf()); // output: [object Object]
trace(myObject.toString()); // output: [object Object]
```

次の例では、ビルトインクラス Date および Array の戻り値を示し、[Object.toString\(\)](#) の戻り値と比較しています。

```
// Create a new Date object set to February 1, 2004, 8:15 AM
// The toString() method returns the current time in human-readable form
// The valueOf() method returns the primitive value in milliseconds
var myDate:Date = new Date(2004,01,01,8,15);
trace(myDate.toString()); // output: Sun Feb 1 08:15:00 GMT-0800 2004
trace(myDate.valueOf()); // output: 1075652100000
```

```
// Create a new Array object containing two simple elements
// In this case both toString() and valueOf() return the same value: one,two
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two
trace(myArray.valueOf()); // output: one,two
```

[toString\(\)](#) を上書きするカスタム クラスの [Object.valueOf\(\)](#) の戻り値の例については、[Object.toString\(\)](#) を参照してください。

関連項目

[toString \(Object.toString メソッド \)](#)

watch (Object.watch メソッド)

```
public watch(name:String, callback:Function, [userData:Object]) : Boolean
```

ActionScript オブジェクトの指定されたプロパティの変更に応じて呼び出されるイベントハンドラを登録します。プロパティが変更されると、イベントハンドラはそれを含むオブジェクト myObject と共に呼び出されます。

callback メソッド定義で return ステートメントを使用して、監視しているプロパティの値に影響を与えることができます。callback メソッドによって返される値は、監視対象のオブジェクトプロパティに割り当てられます。プロパティへの変更を監視するか、変更するか、または禁止するかによって、戻り値の選択は異なります。

- 単にプロパティを監視する場合は、newVal パラメータを返します。
- プロパティの値を変更する場合は、固有の値を返します。
- プロパティを変更しないようにするには、oldVal パラメータを返します。

定義する callback メソッドに return ステートメントがない場合は、監視対象のオブジェクトプロパティに undefined. の値が割り当てられます。

監視ポイントは、変更された newVal (または oldval) を返すことにより、値の代入をフィルタリング (または無効化) できます。監視ポイントが設定されているプロパティを削除しても、その監視ポイントは消えません。後でプロパティを再作成するときに、監視ポイントは依然として有効です。監視ポイントを削除するには、Object.unwatch メソッドを使用します。

1つのプロパティに登録できる監視ポイントは1つのみです。同じプロパティに対して Object.watch() を続けて呼び出すと、元の監視ポイントが置き換えられます。

Object.watch() メソッドの動作は、JavaScript 1.2 以降の Object.watch() 関数に似ています。主な相違点は、userData パラメータです。これは Flash で Object.watch() に追加されたパラメータであり、Netscape Navigator ではサポートされていません。userData パラメータは、イベントハンドラに渡し、イベントハンドラ内で使用できます。

Object.watch() メソッドでは getter/setter プロパティを監視できません。getter/setter プロパティは遅延評価に基づいて処理されます。つまり、プロパティの値は、プロパティが実際に検索されるまで決定されません。遅延評価では、必要になるまで評価が遅延されるため、プロパティの更新頻度が低い場合には便利です。ただし、Object.watch() はプロパティを評価し、callback 関数を呼び出すかどうかを決定する必要があります。getter/setter プロパティを使用すると、Object.watch() はプロパティを常に評価する必要があり、非効率的です。

通常、ActionScript の定義済みプロパティである _x、_y、_width、_height などは getter/setter プロパティであり、Object.watch() では監視できません。

パラメータ

name: `String` - 監視対象であるオブジェクトプロパティの名前を示すストリング。

callback: `Function` - 監視対象のプロパティの変更に応じて呼び出す関数。このパラメータは関数オブジェクトです。ストリングとしての関数名ではありません。callback の形式は `callback(prop, oldVal, newVal, userData)` です。

userData: `Object` (オプション) - callback メソッドに渡す任意の ActionScript データ。userData パラメータを省略すると、`undefined` が callback メソッドに渡されます。

戻り値

`Boolean` - ブール値。監視ポイントが正常に作成された場合は `true` を返します。それ以外の場合は `false` を返します。

例

次の例では、`watch()` を使用し、speed プロパティが速度制限を超えるかどうかを監視しています。

```
// Create a new object
var myObject:Object = new Object();

// Add a property that tracks speed
myObject.speed = 0;

// Write the callback function to be executed if the speed property changes
var speedWatcher:Function = function(prop, oldVal, newVal, speedLimit) {
    // Check whether speed is above the limit
    if (newVal > speedLimit) {
        trace ("You are speeding.");
    }
    else {
        trace ("You are not speeding.");
    }

    // Return the value of newVal.
    return newVal;
}
// Use watch() to register the event handler, passing as parameters:
// - the name of the property to watch: "speed"
// - a reference to the callback function speedWatcher
// - the speedLimit of 55 as the userData parameter
myObject.watch("speed", speedWatcher, 55);

// set the speed property to 54, then to 57
myObject.speed = 54; // output: You are not speeding
myObject.speed = 57; // output: You are speeding

// unwatch the object
myObject.unwatch("speed");
myObject.speed = 54; // there should be no output
```

関連項目

[addProperty \(Object.addProperty メソッド\)](#), [unwatch \(Object.unwatch メソッド\)](#)

security (System.security)

```
Object
|
+-System.security
```

```
public class security
extends Object
```

System.security クラスには、異なるドメインに属する複数の SWF ファイルが互いにどのようにやりとりするかを指定するためのメソッドがあります。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve
プロパティ)
```

メソッド一覧

オプション	シグネチャ	説明
static	<code>allowDomain (domain1:String) : Void</code>	指定されたドメインの SWF ファイルおよび HTML ファイルが、呼び出し元 SWF ファイルのオブジェクトおよび変数、または呼び出し元 SWF ファイルと同じドメイン内にある任意の SWF ファイルのオブジェクトおよび変数にアクセスすることを許可します。
static	<code>allowInsecureDomain (domain:String) : Void</code>	指定されたドメインの SWF ファイルおよび HTML ファイルが、HTTPS プロトコルでホストされた呼び出し元 SWF ファイルのオブジェクトと変数にアクセスすることを許可します。
static	<code>loadPolicyFile(url: String) : Void</code>	url パラメータで指定された場所からドメイン間ポリシー ファイルをロードします。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

allowDomain (security.allowDomain メソッド)

```
public static allowDomain(domain:String) : Void
```

指定されたドメインの SWF ファイルおよび HTML ファイルが、呼び出し元 SWF ファイルのオブジェクトおよび変数、または呼び出し元 SWF ファイルと同じドメイン内にある任意の SWF ファイルのオブジェクトおよび変数にアクセスすることを許可します。

Flash Player 7 以降で再生しているファイルでは、渡されるパラメータはドメイン完全一致規則に従っている必要があります。たとえば、www.domain.com または store.domain.com のいずれかでホストされた SWF ファイルからのアクセスを許可するには、両方のドメイン名を渡す必要があります。

```
// For Flash Player 6
System.security.allowDomain("domain.com");
// Corresponding commands to allow access by SWF files
// that are running in Flash Player 7 or later
System.security.allowDomain("www.domain.com", "store.domain.com");
```

また、ファイルが Flash Player 7 以降で実行されている場合、このメソッドを使用して、セキュアでないプロトコルでホストされている SWF ファイルから、セキュアなプロトコル (HTTPS) を使用してホストされている SWF ファイルへのアクセスを許可することはできません。この場合は、System.security.allowInsecureDomain() を使用してください。

場合によっては、次の状況が発生することがあります。他のドメインから子 SWF ファイルをロードし、その子 SWF ファイルで親 SWF ファイルをスクリプトしたいものの、子 SWF ファイルの最終的なロード元となるドメインがわからないという状況になることがあります。このような状況は、たとえばロードバランシングリダイレクトやサードパーティ製サーバーを使用する場合に発生します。

このような状況では、MovieClip._url プロパティを引数として、このメソッドに使用できます。たとえば、SWF ファイルを my_mc にロードした場合は、System.security.allowDomain(my_mc._url) を呼び出すことができます。

この場合、my_mc の SWF ファイルがロードされるまで待つようにしてください。ファイルが完全にロードされるまで、_url プロパティが最終的な正しい値に設定されないためです。子 SWF ファイルのロードが完了したかどうか確認するには、MovieClipLoader.onLoadComplete を使用するのが最も適しています。

この反対の状況が発生する場合もあります。つまり、親 SWF ファイルで子 SWF ファイルをスクリプトしたいが、親 SWF ファイルのドメインがわからない場合です。こうした状況では、子 SWF ファイルから `System.security.allowDomain(_parent._url)` を呼び出します。この状況では、親 SWF ファイルがロードされるまで待つ必要はありません。親 SWF ファイルは子 SWF ファイルがロードされた時点で既にロードされているからです。

パラメータ

`domain1:String` - `System.Security.allowDomain()` 呼び出しを含む SWF ファイル内のオブジェクトと変数にアクセスできるドメインを指定するストリング。ドメインは、次の形式で指定します。

- "domain.com"
- "http://domain.com"
- "http://IPaddress"

例

`www.macromedia.com/MovieA.swf` の SWF ファイルに次の行が含まれています。

```
System.security.allowDomain("www.shockwave.com");
loadMovie("http://www.shockwave.com/MovieB.swf", my_mc);
```

`MovieA` には `allowDomain()` 呼び出しが含まれているので、`MovieB` は `MovieA` のオブジェクトと変数にアクセスすることができます。`MovieA` にこの呼び出しが含まれていない場合は、Flash のセキュリティ実装により、`MovieB` は `MovieA` のオブジェクトと変数にアクセスできません。

関連項目

[onLoadComplete \(MovieClipLoader.onLoadComplete イベントリスナー\)](#),
[_parent \(MovieClip._parent プロパティ\)](#), [_url \(MovieClip._url プロパティ\)](#),
[allowInsecureDomain \(security.allowInsecureDomain メソッド\)](#)

allowInsecureDomain (security.allowInsecureDomain メソッド)

`public static allowInsecureDomain(domain:String) : Void`

指定されたドメインの SWF ファイルおよび HTML ファイルが、HTTPS プロトコルでホストされた呼び出し元 SWF ファイルのオブジェクトと変数にアクセスすることを許可します。また、指定されたドメインの SWF ファイルが、呼び出し元 SWF ファイルと同じドメイン内にある任意の SWF ファイルにアクセスすることも許可します。

デフォルトでは、HTTPS プロトコルを使用してホストされている SWF ファイルは、HTTPS プロトコルを使用してホストされている SWF ファイルにのみアクセスできます。この実装方法により、HTTPS プロトコルが提供する整合性が保たれます。

このメソッドでデフォルトのビヘイビアを変更することはお勧めできません。デフォルトのビヘイビアを変更すると、HTTPS のセキュリティが損なわれます。ただし、デフォルトのビヘイビアを変更する必要がある場合もあります。たとえば、Flash Player 7 以降用にパブリッシュされた HTTPS ファイルに対して、Flash Player 6 用にパブリッシュされた HTTP ファイルからのアクセスを許可する必要がある場合などです。

Flash Player 6 用にパブリッシュされた SWF ファイルの場合は、`System.security.allowDomain()` を使用して HTTP から HTTPS へのアクセスを許可できます。ただし、Flash Player 7 の場合はセキュリティの実装方法が異なるので、Flash Player 7 以降用にパブリッシュされた SWF ファイルでは、`System.Security.allowInsecureDomain()` を使用して HTTP から HTTPS へのアクセスを許可する必要があります。

メモ : 場合によっては、`System.security.allowInsecureDomain()` 呼び出しが表示される SWF ファイルのドメインと完全に一致する引数で `System.security.allowInsecureDomain()` を呼び出す必要がある場合があります。これは `System.security.allowDomain()` とは異なります。

`System.security.allowDomain()` の場合は、SWF ファイル自身のドメインを引数として使用して呼び出しを行う必要がありません。`System.security.allowInsecureDomain()` でこの操作が必要になるのは、ドメインが同一である場合でも、デフォルトでは、`http://foo.com` の SWF ファイルは `https://foo.com` の SWF ファイルをスクリプティングできないからです。

パラメータ

`domain: String` - `www.myDomainName.com` や `store.myDomainName.com` などの完全なドメイン名。

例

次の例では、登録済みの学生だけがアクセスできるよう、セキュアなドメイン上で数学のテストをホストしています。また、特定の概念を説明する複数の SWF ファイルを作成し、これをセキュアではないドメインでホストしているとします。ここで、学生が、概念情報を含む SWF ファイルからテストにアクセスできるようにしてみましょう。

```
// This SWF file is at https://myEducationSite.somewhere.com/mathTest.swf  
  
// Concept files are at http://myEducationSite.somewhere.com  
  
System.security.allowInsecureDomain("myEducationSite.somewhere.com");
```

関連項目

[allowDomain \(security.allowDomain メソッド\)](#)

loadPolicyFile (security.loadPolicyFile メソッド)

```
public static loadPolicyFile(url:String) : Void
```

url パラメータで指定された場所からドメイン間ポリシーファイルをロードします。Flash Player では、Flash ミーティングが置かれているサーバー以外のサーバーからデータをロードすることを許可する許可メカニズムとして、ポリシーファイルが使用されます。

Flash Player 7.0.14.0 では、ポリシーファイルの検索が行われるのはデータロード要求が行われたサーバーの /crossdomain.xml に限られます。XMLSocket 接続試行の場合、XMLSocket 接続の試行が行われたサブドメインのポート 80 上にある HTTP サーバーで、/crossdomain.xml に検索が行われます。さらに Flash Player 7.0.14.0 (およびそれ以前のバージョン) では、XMLSocket 接続がポート 1024 以上に限られています。

System.security.loadPolicyFile() を追加することで、Flash Player 7.0.19.0 では任意の場所からポリシーファイルをロードできるようになります。次に例を示します。

```
System.security.loadPolicyFile("http://foo.com/sub/dir/pf.xml");
```

これにより、Flash Player は指定された URL からポリシーファイルを取得できるようになります。この場所に置かれているポリシーファイルによって得られる許可は、サーバーの仮想ディレクトリ階層で同レベル以下のコンテンツすべてに適用されます。次に示すコードは、前の例の続きです。

```
loadVariables("http://foo.com/sub/dir/vars.txt") // allowed  
loadVariables("http://foo.com/sub/dir/deep/vars2.txt") // allowed  
loadVariables("http://foo.com/elsewhere/vars3.txt") // not allowed
```

loadPolicyFile() を使用して、任意の数のポリシーファイルをロードできます。ポリシーファイルを必要とする要求がある場合、Flash Player はポリシーファイルのダウンロードがすべて完了するまで必ず待機します。その間に要求が拒否されることはありません。loadPolicyFile() で指定されたポリシーファイルによって要求が許可されなかった場合は、最終的にデフォルトの場所である /crossdomain.xml が参照されます。

特定のポート番号で xmlsocket プロトコルを使用することで、直接 XMLSocket サーバーからポリシーファイルを取得することができます。次に例を示します。

```
System.security.loadPolicyFile("xmlsocket://foo.com:414");
```

このコードを使用すると、Flash Player は指定されたホストとポートからポリシーファイルを取得しようとします。1024 以上のポートだけでなく、任意のポートを使用できます。指定されたポートを使用して接続が確立されると、Flash Player は null バイトで終了する <policy-file-request /> を送信します。XMLSocket サーバーを設定することで、ポリシーファイルと通常の XMLSocket 接続に同じポートを使用することができます。この場合、サーバーは <policy-file-request /> が受信されるまで待機し、その後でポリシーファイルを送信する必要があります。標準の接続とは異なるポートを使用して、ポリシーファイルを提供するようにサーバーを設定することもできます。この場合は、ポリシーファイル専用のポートで接続が確立されるとすぐにポリシーファイルを送信できます。ポリシーファイルを終了するためにサーバーから null バイトを送信し、それ以降の接続を閉じる必要があります。サーバー側で接続を閉じなければ、最後の null バイトが受信されるとすぐに Flash Player 側で接続が閉じられます。

XMLSocket サーバーで提供するポリシーファイルのシンタックスは、他のポリシーファイルとほぼ同じですが、アクセスを許可するポートも指定する必要がある点が異なります。ポリシーファイルが 1024 未満のポートから提供される場合、任意のポートへのアクセスが許可されます。ポリシーファイルが 1024 以上のポートから提供される場合、他の 1024 以上のポートにのみアクセスを許可できます。許可するポートは、<allow-access-from> タグの "to-ports" 属性で指定します。単一のポート番号、ポート範囲、ワイルドカードを使用できます。次に、XMLSocket ポリシーファイルの例を示します。

```
<cross-domain-policy>

<allow-access-from domain="*" to-ports="507" />

<allow-access-from domain="*.foo.com" to-ports="507,516" />

<allow-access-from domain="*.bar.com" to-ports="516-523" />

<allow-access-from domain="www.foo.com" to-ports="507,516-523" />

<allow-access-from domain="www.bar.com" to-ports="*" />

</cross-domain-policy>
```

元のデフォルトの場所、つまり HTTP サーバーのポート 80 の /crossdomain.xml から取得されたポリシーファイルでは、1024 以上の全ポートへのアクセスが暗黙的に許可されています。HTTP サーバーの他の場所から、XMLSocket 操作を許可するポリシーファイルを取得することはできません。XMLSocket ポリシーファイルを独自の場所に置く場合、XMLSocket サーバー上に置く必要があります。

1024 未満のポートに接続する機能は新しいため、ムービークリップが自分自身のサブドメインに接続する場合でも、loadPolicyFile() によってロードするポリシーファイルを使用して常にこの接続の許可を行う必要があります。

パラメータ

url:String - ロードするドメイン間ポリシーファイルが置かれている URL を表すストリング。

Selection

```
Object
 |
 +-Selection
```

```
public class Selection
extends Object
```

Selection クラスを使用すると、ムービー内でカーソルを置くテキストフィールド、つまりフォーカスが置かれているフィールドを設定および制御することができます。選択範囲のインデックスはゼロから始まります。つまり、第 1 の位置は 0、第 2 の位置は 1、以下同様です。

フォーカスのあるフィールドは一度に 1 つしか存在しないため、Selection クラスにはコンストラクタ関数がありません。

Selection オブジェクトは、デバイスがオンラインテキスト入力をサポートしている場合のみ有効です。デバイスがオンラインテキスト入力をサポートしておらず、FEP(Front-End Processor : 前置プロセッサ)を使用してテキストを入力する場合、Selection オブジェクトの呼び出しはすべて無視されます。

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve
プロパティ)
```

イベント一覧

イベント	説明
onSetFocus = function([oldfocus], [newfocus]) {}	フォーカスが変更されると、通知されます。

メソッド一覧

オプション	シグネチャ	説明
static	<code>addListener(listener: Object) : Void</code>	フォーカスの変更通知を受け取るオブジェクトを登録します。
static	<code>getFocus() : String</code>	フォーカスのあるオブジェクトのターゲットパスを指定するストリングを返します。
static	<code>removeListener(listener:Object) : Boolean</code>	<code>Selection.addListener()</code> メソッドを使用して以前に登録したオブジェクトを削除します。
static	<code>setFocus(newFocus: Object) : Boolean</code>	<code>newFocus</code> パラメータで指定する選択可能(編集可能)なテキストフィールド、ボタン、またはムービークリップにフォーカスを設定します。
static	<code>setSelection(beginIndex:Number, endIndex:Number) : Void</code>	現在フォーカスのあるテキストフィールドの選択範囲を設定します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (Selection.addListener メソッド)

`public static addListener(listener:Object) : Void`

フォーカスの変更通知を受け取るオブジェクトを登録します。フォーカスが変わると (`Selection.setFocus()` メソッドが呼び出された場合など)、`addListener()` で登録されたすべてのリスナーオブジェクトの `onSetFocus()` メソッドが呼び出されます。複数のオブジェクトが フォーカスの変更通知を監視できます。指定したリスナーが登録済みである場合、変化は起きません。

パラメータ

`listener:Object` - `onSetFocus` メソッドを持つ新しいオブジェクト。

例

次の例では、実行時に 2 つのテキスト入力フィールドを作成し、各テキストフィールドの境界線を true に設定します。このコードでは、focusListener という名前の ActionScript オブジェクト(汎用オブジェクト)を新しく作成しています。そして、このオブジェクト自体の onSetFocus プロパティを定義し、関数を割り当てています。この関数は 2 つのパラメータを取ります。フォーカスを失ったテキストフィールドへの参照と、フォーカスを受け取ったテキストフィールドへの参照です。関数では、フォーカスを失ったテキストフィールドの border プロパティを false に、フォーカスを受け取ったテキストフィールドの border プロパティを true に、それぞれ設定しています。

```
this.createTextField("one_txt", 99, 10, 10, 200, 20);
this.createTextField("two_txt", 100, 10, 50, 200, 20);
one_txt.border = true;
one_txt.type = "input";
two_txt.border = true;
two_txt.type = "input";

var focusListener:Object = new Object();
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
    oldFocus_txt.border = false;
    newFocus_txt.border = true;
};
Selection.addListener(focusListener);
```

関連項目

[setFocus \(Selection.setFocus メソッド\)](#)

getFocus (Selection.getFocus メソッド)

public static getFocus() : String

フォーカスのあるオブジェクトのターゲットパスを指定するストリングを返します。

- TextField オブジェクトにフォーカスがあり、インスタンス名が付いている場合、getFocus() メソッドは TextField オブジェクトのターゲットパスを返します。それ以外は、TextField の変数名を返します。
- Button オブジェクトまたはボタンムービークリップにフォーカスがある場合、getFocus() メソッドはそれらのターゲットパスを返します。
- TextField オブジェクト、Button オブジェクト、コンポーネントインスタンス、ボタンムービークリップのいずれにもフォーカスがない場合は、getFocus() メソッドは null を返します。

戻り値

[String](#) - ストリングまたは null。

例

次の例では、現在フォーカスされているオブジェクトのパスを出力するテキストフィールドを作成します。次に、インターバル関数を使用してフィールドを定期的に更新します。これをテストするには、インスタンスにいくつかのボタンを追加して、異なるインスタンス名を付けます。次に、使用する AS ファイルまたは FLA ファイルに次の ActionScript を追加します。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 150, 25);

function FocusUpdate()
{
    s = Selection.getFocus();
    if ( s )
    {
        status_txt.text = s;
    }
}

setInterval( FocusUpdate, 100 );
```

関連項目

[onSetFocus \(Selection.onSetFocus イベントリスナー\)](#),
[setFocus \(Selection.setFocus メソッド\)](#)

onSetFocus (Selection.onSetFocus イベントリスナー)

onSetFocus = function([oldfocus], [newfocus]) {}

フォーカスが変更されると、通知されます。このリスナーを使用するには、リスナーオブジェクトを作成します。次にこのリスナーの関数を定義し、Selection.addListener() メソッドを使用してリスナーを Selection オブジェクトに登録します。次に例を示します。

```
var someListener:Object = new Object();
someListener.onSetFocus = function () {
    // statements
}
Selection.addListener(someListener);
```

1つのイベントに関する通知を複数のリスナーで受け取ることができるので、リスナーごとに異なる複数のコードを作成して連携させることもできます。

パラメータ

oldfocus: (オプション)- フォーカスを失うオブジェクト。

newfocus: (オプション)- フォーカスを受け取るオブジェクト。

例

次の例では、ダイナミックに作成された複数のテキストフィールドの間で、SWF ファイルの入力フォーカスの移動のタイミングを判断する方法について説明します。次の ActionScript を FLA または AS ファイルに入力し、ドキュメントをテストします。

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
    if (this[i] instanceof TextField) {
        this[i].border = true;
        this[i].type = "input";
    }
}

this.createTextField("status_txt", this.getNextHighestDepth(), 200, 10, 300,
    100);
status_txt.html = true;
status_txt.multiline = true;

var someListener:Object = new Object();
someListener.onSetFocus = function(oldFocus, newFocus) {
    status_txt.htmlText = "<b>setFocus triggered</b>";
    status_txt.htmlText += "<textformat tabStops='[20,80]'>";
    status_txt.htmlText += "&nbsp;\toldFocus:\t"+oldFocus;
    status_txt.htmlText += "&nbsp;\tnewFocus:\t"+newFocus;
    status_txt.htmlText += "&nbsp;\tgetFocus:\t"+Selection.getFocus();
    status_txt.htmlText += "</textformat>";
};
Selection.addListener(someListener);
```

関連項目

[addListener \(Selection.addListener メソッド\)](#), [setFocus \(Selection.setFocus メソッド\)](#)

removeListener (Selection.removeListener メソッド)

public static removeListener(listener:[Object](#)) : Boolean

Selection.addListener() メソッドを使用して以前に登録したオブジェクトを削除します。

パラメータ

listener:[Object](#) - フォーカスの通知受信を中止するオブジェクト。

戻り値

[Boolean](#) - listener オブジェクトが正常に削除された場合は true を返します。listener オブジェクトが正常に削除されなかった場合 (listener が Selection オブジェクトのリスナーリストにない場合など) は false を返します。

例

次の ActionScript は、いくつかのテキストフィールドインスタンスをダイナミックに作成します。テキストフィールドを選択すると、[出力] パネルに情報が表示されます。remove_btn インスタンスをクリックすると、リスナーが削除され、[出力] パネルに情報が表示されなくなります。

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
    if (this[i] instanceof TextField) {
        this[i].border = true;
        this[i].type = "input";
    }
}

var selectionListener:Object = new Object();
selectionListener.onSetFocus = function(oldFocus, newFocus) {
    trace("Focus shifted from "+oldFocus+" to "+newFocus);
};
Selection.addListener(selectionListener);

remove_btn.onRelease = function() {
    trace("removeListener invoked");
    Selection.removeListener(selectionListener);
};
```

関連項目

[addListener \(Selection.addListener メソッド \)](#)

setFocus (Selection.setFocus メソッド)

```
public static setFocus(newFocus:Object) : Boolean
```

newFocus パラメータで指定する選択可能(編集可能)なテキストフィールド、ボタン、またはムービークリップにフォーカスを設定します。ドットまたはスラッシュ表記を使用してパスを指定できます。相対パスと絶対パスのいずれでも使用できます。ActionScript 2.0 の場合は、ドット表記を使用する必要があります。

null を渡すと、現在のフォーカスは削除されます。

パラメータ

newFocus: Object - ボタン、ムービークリップ、テキストフィールドの各インスタンスなどのオブジェクト、またはボタン、ムービークリップ、テキストフィールドの各インスタンスへのパスを指定するストリング。

戻り値

Boolean - ブール値。フォーカスの試行が成功した場合は true、失敗した場合は false を返します。

例

次の例では、ブラウザウィンドウで実行中に、テキストフィールドは username_txt テキストフィールドにフォーカスを設定します。必要なテキストフィールド(username_txt と password_txt)のいずれかが未入力である場合、カーソルはデータが未入力のテキストフィールドに自動的にフォーカスを与えます。たとえば、ユーザーが username_txt テキストフィールドに何も入力せずに送信ボタンを押すと、エラーメッセージが表示され、カーソルは username_txt テキストフィールドにフォーカスを与えます。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 100, 70, 100, 22);
this.createTextField("username_txt", this.getNextHighestDepth(), 100, 100, 100, 22);
this.createTextField("password_txt", this.getNextHighestDepth(), 100, 130, 100, 22);
this.createEmptyMovieClip("submit_mc", this.getNextHighestDepth());
submit_mc.createTextField("submit_txt", this.getNextHighestDepth(), 100, 160, 100, 22);
submit_mc.submit_txt.autoSize = "center";
submit_mc.submit_txt.text = "Submit";
submit_mc.submit_txt.border = true;
submit_mc.onRelease = checkForm;
username_txt.border = true;
password_txt.border = true;
username_txt.type = "input";
password_txt.type = "input";
password_txt.password = true;
Selection.setFocus("username_txt");
fscommand("activateTextField");
```

```
//  
function checkForm():Boolean {  
    if (username_txt.text.length == 0) {  
        status_txt.text = "fill in username";  
        Selection.setFocus("username_txt");  
        fscommand("activateTextField");  
        return false;  
    }  
    if (password_txt.text.length == 0) {  
        status_txt.text = "fill in password";  
        Selection.setFocus("password_txt");  
        fscommand("activateTextField");  
        return false;  
    }  
    status_txt.text = "success!";  
    Selection.setFocus(null);  
    return true;  
}
```

関連項目

[getFocus \(Selection.setFocus メソッド\)](#)

setSelection (Selection.setSelection メソッド)

`public static setSelection(beginIndex:Number, endIndex:Number) : Void`

現在フォーカスのあるテキストフィールドの選択範囲を設定します。新しい選択範囲は、beginIndex パラメータで指定されたインデックスから始まり、endIndex パラメータで指定されたインデックスで終わります。選択範囲のインデックスはゼロから始まります。つまり、第1の位置は0、第2の位置は1、以下同様です。現在フォーカスされているテキストフィールドがない場合は、このメソッドは何も実行しません。`setSelection()` メソッドを呼び出し、テキストコントロールをフォーカスしたとき、選択状態のハイライトは、テキストフィールドがアクティブな状態で編集されている場合にのみ表示されます。`setSelection()` メソッドは、`Selection.setFocus()` の後、または `onSetFocus()` イベントハンドラ内から呼び出すことができます。ただし、選択内容は、`fscommand activateTextField` コマンドを呼び出した後にのみ表示されます。

パラメータ

`beginIndex:Number` - 選択範囲の始めのインデックス。

`endIndex:Number` - 選択範囲の終わりのインデックス。

例

次の ActionScript コードは、実行時にテキストフィールドを作成し、ストリングを追加します。次に onSetFocus イベントにイベントハンドラを割り当てます。このイベントは、テキストフィールド内のすべてのテキストを選択して、編集セッションをアクティブにします。

メモ: Selection.setSelection() メソッドが呼び出された場合は、fscommand activateTextField コマンドが呼び出されてテキストフィールドがアクティブになるまで、テキストはスクリーンに表示されません。

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.type = "input";
myText_txt.text = "this is my text";
myText_txt.onSetFocus = function(){
    Selection.setSelection(0,myText_txt.text.length);
    fscommand("activateTextField");
}
```

次の例では、endIndex パラメータを指定しない方法を示します。最初の文字を選択するには、値が 0 でなく 1 の endIndex を使用する必要があります。endIndex パラメータを 0 に設定すると、何も選択されません。

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.text = "this is my text";
this.onEnterFrame = function () {
    Selection.setFocus("myText_txt");
    Selection.setSelection(0, 1);
    delete this.onEnterFrame;
}
```

SharedObject

```
Object
 |
 +-SharedObject
```

```
public dynamic class SharedObject
extends Object
```

Flash Lite バージョンの SharedObject クラスを使用すると、Flash SWF ファイルを閉じるときにデータをデバイスに保存し、再び再生したときにデバイスからそのデータをロードすることができます。Flash Lite 共有オブジェクトは、名前と値のペアのセットをデバイスに保存します。

メモ: "SharedObject" という名前は、Flash の SharedObject クラスから派生しています。Flash バージョンのこのクラスを使用すると、保存されているデータを複数の Flash SWF ファイルで共有できます。ただし、Flash Lite バージョンの SharedObject クラスは、異なる Flash SWF ファイル間でのデータの共有をサポートしていません。

Flash Lite では、元のバージョンから修正された SWF ファイルは、同じ名前であっても異なるバージョンであると見なされます。これは Flash Player の動作と異なります。Flash Player では、SWF ファイルが修正されても、その URL と名前が同じである場合、同じファイルであると見なされます。Flash Lite では、2 つの異なるバージョンの SWF ファイルは、お互いの共有オブジェクトにアクセスすることはできません。

Flash プラットフォームでの一貫性を保つため、Flash Lite プレーヤーには同じ ActionScript 構成と呼び出し規則が使用されます。

次の例では、共有オブジェクトの使用例を示します。

- ユーザーが中古車の一覧を検索できるようにするサービスで、ユーザーインターフェイスとして Flash アプリケーションを使用できます。このアプリケーションは、ユーザーが入力する検索語と設定に基づいて、中古車の一覧を提供するサーバーに接続します。ユーザーが最後に行った検索を保存し、次回にこの SWF ファイルを再生したときに、フォームにデータを表示することができます。そのためには、ユーザーが新しい検索を実行するたびに、検索パラメータを保存する SharedObject インスタンスを作成します。SWF ファイルが閉じられると、共有オブジェクト内のデータがデバイスに保存されます。次回に SWF ファイルを再生すると、共有オブジェクトがロードされ、ユーザーが前回に入力したのと同じ検索データがフォームに表示されます。
- また、ユーザーが音楽のレビューを検索できるようにするサービスで、ユーザーインターフェイスとして Flash アプリケーションを使用できます。このアプリケーションでは、ユーザーのお気に入りのアルバムに関する情報を保存することができます。この情報はリモートサービスに保存できますが、アプリケーションがサービスに接続できない場合に問題が発生します。また、リモートサービスからのデータの取得に時間がかかり、ユーザー エクスペリエンスが損なわれる可能性もあります。共有オブジェクトを使用することにより、アプリケーションはアルバムに関する情報をデバイスに保存し、必要なときにすばやくロードできます。

メモ：モバイルデバイスでは容量に制限があるため、データは完全に一貫したものにはなりません。状況によっては、最も古いデータがデバイスから削除されるプラットフォームもあります。

ローカル共有オブジェクトを作成するには、次のシンタックスを使用します。

```
var so:shared object = shared object.getLocal("mySharedObject");
```

ハンドセットでのデータの読み取りと書き込みは低速な場合があります。アプリケーションが要求したときに、データをすぐにデバイスから利用できるようにするために、Flash Lite 2.0 では、リスナーを設定する必要があります。デバイスが共有オブジェクトのデータをロードすると、プレーヤーによりリスナーが呼び出されます。getLocal() の呼び出しで返される SharedObject インスタンスにアクセスするメソッドは、リスナーが呼び出されるのを待ってから処理を実行する必要があります。

例

次の例では、SWF ファイルが Prefs というリスナー関数を作成し、共有オブジェクトを作成します。データが利用可能になると、プレーヤーは loadCompletePrefs 関数を呼び出します。

```
function loadCompletePrefs (mySO:SharedObject) {
    if (0 == mySO.getSize() ) {
    }
    // If the size is 0, we need to initialize the data:
    mySO.data.name = "Sigismund";
    mySO.data.email = "siggy@macromedia.com";
}
else
{
    // Trace all the data in mySO:
    trace( "Prefs:" );
    for (var idx in mySO.data) {
        trace( " " + idx +": " + mySO.data[idx] );
    }
}
}

SharedObject.addListener( "Prefs", loadCompletePrefs );
```

```
// We can now create the shared object:
var Prefs:SharedObject = SharedObject.getLocal("Prefs");
```

データが利用可能であることをプレーヤーがリスナーに通知すると、アプリケーションは getLocal() メソッドの呼び出しで返される共有オブジェクトを、Flash で共有オブジェクトを使用するのと同じ方法で使用することができます。コンテンツの再生中、アプリケーションはプロパティを追加、修正、または削除できます。コンテンツをアップロードするときに、共有オブジェクトをデバイスに書き込むことができます。ただし、共有オブジェクトが確実にデバイスに書き込まれるようになるためには、アプリケーションは flush() メソッドを呼び出して書き込み処理を強制する必要があります。

Flash Lite 共有オブジェクトは、ローカルに保存された SWF ファイルのみが利用できます。ネットワーク対応ブラウザで再生される SWF ファイルは、Flash Lite 共有オブジェクトを使用することはできません。

SWF ファイルごとの Flash Lite 共有オブジェクトの総保存容量は、デバイスの既定のサイズによって制限されます。このサイズは、SharedObject.getMaxSize() メソッドを使用して決定できます。

メモ : Flash Lite 2.0 では、リモート共有オブジェクトはサポートされません。

関連項目

[flush \(SharedObject.flush メソッド\)](#), [onStatus \(SharedObject.onStatus ハンドラ\)](#)

プロパティ一覧

オプション	プロパティ	説明
	<code>data:Object</code>	オブジェクトの <code>data</code> プロパティに割り当てられた属性のコレクション。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

イベント一覧

イベント	説明
<code>onStatus = function (infoObject:Object) {}</code>	共有オブジェクトに対してエラー、警告、情報通知が送信されたときに呼び出されます。

メソッド一覧

オプション	シグネチャ	説明
static	<code>addListener (objectName:String, notifyFunction: Function) : Void</code>	Flash Lite プレーヤーがデバイスから共有オブジェクトデータをロードしたときに、プレーヤーが呼び出すイベントリスナーを作成します。
	<code>clear() : Void</code>	共有オブジェクトのすべてのデータを消去し、ディスクから共有オブジェクトを削除します。
	<code>flush (minDiskSpace: Number) : Object</code>	永続ローカルファイルに共有オブジェクトを書き込みます。
static	<code>getLocal (name:String) : SharedObject</code>	現在のクライアントだけが利用できるローカル永続共有オブジェクトへの参照を返します。
static	<code>getMaxSize() : Number</code>	モバイル共有オブジェクトをデバイスに保存するためには SWF ファイルが使用できる総バイト数を返します。
	<code>getSize() : Number</code>	共有オブジェクトの現在のサイズ(バイト数)を取得します。
static	<code>removeListener (objectName:String)</code>	<code>addListener()</code> メソッドを使用して追加されたリスナーを削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (SharedObject.addListener メソッド)

public static addListener(objectName:String, notifyFunction:Function) : Void
Flash Lite プレーヤーがデバイスから共有オブジェクトデータをロードしたときに、プレーヤーが呼び出すイベントリスナーを作成します。getLocal() メソッドの呼び出しで返される SharedObject インスタンスにアクセスするメソッドは、この関数が呼び出されるのを待ってから処理を実行する必要があります。

パラメータ

objectName:String - 共有オブジェクトの名前を表すストリング。

notifyFunction:Function - getLocal() メソッドが実行され、データのロードが終了したことをアプリケーションに通知するためにプレーヤーが呼び出す関数の名前。

clear (SharedObject.clear メソッド)

public clear() : Void

共有オブジェクトのすべてのデータを消去し、ディスクから共有オブジェクトを削除します。my_so への参照はアクティブなままで、my_so は空になります。

例

次の例では、共有オブジェクトにデータを設定してから、その共有オブジェクトのすべてのデータを空にします。

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");  
my_so.data.name = "Hector";  
trace("before my_so.clear():");  
for (var prop in my_so.data) {  
    trace("\t"+prop);  
}  
trace("");  
my_so.clear();  
trace("after my_so.clear():");  
for (var prop in my_so.data) {
```

```
    trace("\t"+prop);
}

この ActionScript は、[ 出力 ] パネルに次のメッセージを表示します。

before my_so.clear():
    name

after my_so.clear():
```

data (SharedObject.data プロパティ)

```
public data : Object
```

オブジェクトの data プロパティに割り当てられた属性のコレクション。それぞれの属性は、ActionScript または JavaScript の基本タイプのオブジェクトです。たとえば、Array、Number、Boolean などです。次のコードでは、共有オブジェクトにさまざまな値を割り当てています。

メモ: Flash Lite では、共有オブジェクトリスナーが呼び出されていない場合、data プロパティに未定義の値が含まれることがあります。詳細については、addListener() メソッドの説明を参照してください。

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

for (var prop in my_so.data) {
    trace(prop+": "+my_so.data[prop]);
}

soResult = "";
for (var prop in my_so.data) {
    soResult += prop+": "+my_so.data[prop] +"\n";
}
result.text = soResult;
```

永続的なオブジェクトの場合は、共有オブジェクトの data プロパティのすべての属性が保存されます。また共有オブジェクトには次の情報が含まれています。

```
userName: Ramona
adminPrivileges: true
itemNumbers: 101,346,483
```

メモ: so.data = someValue のように、data プロパティに直接値を割り当てないでください。このような割り当ては無視されます。

ローカル共有オブジェクトの属性を削除するには、`delete so.data.attributeName` というコードを使用します。ローカル共有オブジェクトの属性を `null` または `undefined` に設定しても属性は削除されません。

共有オブジェクトのプライベート値（オブジェクトの使用中にそのクライアントインスタンスでのみ利用でき、閉じるときにオブジェクトと共に保存されない値）を作成するには、`data` 以外の名前のプロパティを作成して、その値を保存します。次に例を示します。

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.favoriteColor = "blue";
my_so.favoriteNightClub = "The Bluenote Tavern";
my_so.favoriteSong = "My World is Blue";

for (var prop in my_so) {
    trace(prop+": "+my_so[prop]);
}
```

共有オブジェクトには次のデータが含まれます。

```
favoriteSong: My World is Blue
favoriteNightClub: The Bluenote Tavern
favoriteColor: blue
data: [object Object]
```

例

次の例では、`my_so` という共有オブジェクトにテキストを保存します（完全な例については `SharedObject.getLocal()` を参照）。

```
var my_so:SharedObject = SharedObject.getLocal("savedText");

// myText is an input text field and inputText is a dynamic text field.
myText.text = my_so.data.myTextSaved;
// Assign an empty string to myText_ti if the shared object is undefined
// to prevent the text input box from displaying "undefined" when
// this script is first run.
if (myText.text == "undefined") {
    myText.text = "";
}

changedListener = new Object();
changedListener.onChanged = function (changedField) {
    my_so.data.myTextSaved = changedField.text;
    inputText.text = "";
    inputText.text = my_so.data.myTextSaved;
}
myText.addListener(changedListener);
```

flush (SharedObject.flush メソッド)

public flush(minDiskSpace:Number) : Object

永続ローカルファイルに共有オブジェクトを書き込みます。共有オブジェクトが確実にデバイスに書き込まれるようにするには、アプリケーションは flush() メソッドを呼び出して書き込み処理を強制する必要があります。

Flash Player とは異なり、書き込み処理は非同期であり、結果はすぐに使用できません。

パラメータ

minDiskSpace:Number - このオブジェクトに割り当てる必要のあるバイト数を示す整数。デフォルト値は 0 です。

戻り値

Object - true または false の布尔値。または "pending" のストリング値。flush() メソッドはほとんどの要求に対して pending を返しますが、次の例外があります。

- データを書き込む必要がない(データが既に書き込まれている)場合、flush() は true を返します。
- minimumDiskSpace パラメータが SWF ファイルで利用できる最大容量または空き容量を超える場合や、要求の処理中にエラーが発生した場合、flush() は false を返します。

flush() メソッドが pending を返す場合、Flash Lite プレーヤーでは、共有オブジェクトで利用できるディスク容量を増やすために、空き容量を作るようユーザーに要求するダイアログボックスを表示することができます。将来的に共有オブジェクトを保存する際に、領域が自動的に拡張されるようにする(pending が返されないようにする)には、minimumDiskSpace に値を指定します。ファイルにオブジェクトを書き込む際には、現在のサイズの共有オブジェクトを保存するために十分な領域だけではなく、minimumDiskSpace に指定したバイト数が確認されます。

例

次の例では、flush() メソッドの戻り値を処理しています。

```
so_big = SharedObject.getLocal("large");

so_big.data.name = "This is a long string of text.";
so_big.flush();
var flushResult = so_big.flush();

switch (flushResult) {
case 'pending' :
    result.text += "pending";
    break;
case true :
    result.text += "Data was flushed.";
    break;
case false :
```

```
    result.text += "Test failed. Data was not flushed.";
    break;
}
```

関連項目

[clear \(SharedObject.clear メソッド\)](#), [onStatus \(SharedObject.onStatus ハンドラ\)](#)

getLocal (SharedObject.getLocal メソッド)

`public static getLocal(name:String) : SharedObject`

現在のクライアントだけが利用できるローカル永続共有オブジェクトへの参照を返します。共有オブジェクトがまだ存在しない場合は、`getLocal()` により作成されます。このメソッドは、`SharedObject` クラスの静的メソッドです。

メモ : Flash Lite では、2つの SWF ファイル間で共有オブジェクトを共有することはできません。

オブジェクトを変数に代入するには、次のようなシンタックスを使用します。

```
var so:SharedObject = SharedObject.getLocal("savedData")
```

データはデバイスで読み取るためにすぐに利用できない場合があるので、アプリケーションは `name` で識別される共有オブジェクトのリスナーを作成および登録する必要があります。詳細については、`addListener()` メソッドの説明を参照してください。

パラメータ

`name:String` - オブジェクトの名前を表すストリング。名前にはスラッシュ (/) を含めます。たとえば、`work/addresses` は有効な名前です。共有オブジェクト名にスペース、および以下の文字を含めることはできません。

~ % & \ ; : " ' , < > ? #

戻り値

`SharedObject` - ローカルに永続化され、現在のクライアントでのみ利用できる共有オブジェクトへの参照が返されます。共有オブジェクトを作成または見つけられない場合、`getLocal()` は `null` を返します。

デバイスにより、サードパーティの Flash コンテンツによる永続共有オブジェクトの作成と保存が禁止されている場合、このメソッドは失敗し、`null` を返します。

例

次の例では、再生した最後のフレームをローカル共有オブジェクト kookie に保存しています。

```
// Get the kookie
var my_so:SharedObject = SharedObject.getLocal("kookie");

// Get the user of the kookie and go to the frame number saved for this user.
if (my_so.data.user != undefined) {
    this.user = my_so.data.user;
    this.gotoAndStop(my_so.data.frame);
}
```

SWF ファイルの各フレームに次のコードを挿入します。

```
// On each frame, call the rememberme function to save the frame number.
function rememberme() {
    my_so.data.frame=this._currentframe;
    my_so.data.user="John";
}
```

getMaxSize (SharedObject.getMaxSize メソッド)

public static getMaxSize() : Number

モバイル共有オブジェクトをデバイスに保存するために SWF ファイルが使用できる総バイト数を返します。

たとえば、このメソッドが 1K を返す場合、ムービーは 1K の 1 つの共有オブジェクトを保存できます。または、合計サイズが 1K を超えない限り、それより小さい複数の共有オブジェクトを保存できます。このメソッドは、SharedObject クラスの静的メソッドです。

戻り値

Number - デバイスでムービーの保存が許可される総バイト数を指定する数値。これは、loadMovie() を使用して動的にロードされるすべてのコンテンツで利用できるサイズでもあります。

例

次の例では、Flash Lite 共有オブジェクトを作成する前に、1KB 以上の容量があるかどうかを確認します。

```
if (SharedObject.getMaxSize() > 1024) {
    var my_so:SharedObject = SharedObject.getLocal("sharedObject1");
} else {
    trace("SharedObject's maximum size is less than 1 KB.");
}
```

getSize (SharedObject.getSize メソッド)

```
public getSize():Number
```

共有オブジェクトの現在のサイズ(バイト数)を取得します。

すべてのデータプロパティを順に確認することによって、共有オブジェクトのサイズが計算されます。オブジェクトが持つデータプロパティが多いほど、サイズの計算に時間がかかります。オブジェクトのサイズを調べる処理は、非常に時間がかかる場合があり、特に必要がない限り、このメソッドの使用を避けることができます。

共有オブジェクトリスナーが呼び出されていない場合、`getSize()` は 0 を返します。リスナーの使用の詳細については、`addListener()` メソッドを参照してください。

戻り値

`Number` - 共有オブジェクトのサイズ(バイト数)を示す数値。

例

次の例では、共有オブジェクト `my_so` のサイズを取得します。

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

var soSize:Number = my_so.getSize();
trace(soSize);
```

onStatus (SharedObject.onStatus ハンドラ)

```
onStatus = function(infoObject:Object) {}
```

共有オブジェクトに対してエラー、警告、情報通知が送信されたときに呼び出されます。このイベントハンドラに応答するには、共有オブジェクトによって生成される情報オブジェクトを処理する関数を作成する必要があります。

情報オブジェクトには、`onStatus` ハンドラの結果ストリングを含む `code` プロパティと、"Status" または "Error" のいずれかのストリングを含む `level` プロパティがあります。

このハンドラ以外に、Flash Lite には `System.onStatus` というスーパー関数もあります。特定のオブジェクトに対して `onStatus` が呼び出され、それに応答する関数が割り当てられていない場合、`System.onStatus` に割り当てられた関数があれば、この関数が実行されます。

特定の SharedObject アクティビティが発生した場合、次のイベントによって通知されます。

code プロパティ	level プロパティ	説明
SharedObject.Flush.Failed	Error	"pending" を返した SharedObject.flush() メソッドが失敗しました。[ローカル記憶領域] ダイアログ ボックスが表示が表示されたときに、ユーザーが、共有オブジェクトに対して追加のディスク領域を割り当てませんでした。
SharedObject.Flush.Success	Status	"pending" を返した SharedObject.flush() コマンドが正常に完了しました。ユーザーは、共有オブジェクトに対して追加のディスク領域を割り当てました。

パラメータ

infoObject:Object - ステータスマッセージに従って定義されるパラメータ。

例

次の例では、ユーザーが SharedObject インスタンスのディスクへの書き込みを許可したかどうかに応じて、異なるメッセージを表示します。

```
this.createTextField("message_txt", this.getNextHighestDepth(), 0, 30, 120, 50);
this.message_txt.wordWrap = true;

this.createTextField("status_txt", this.getNextHighestDepth(), 0, 90, 120, 100);
this.status_txt.wordWrap = true;

var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

my_so.onStatus = function(infoObject:Object) {
    for (var i in infoObject) {
        status_txt.text += i + "-" + infoObject[i] + "\n";
    }
};

var flushResult = my_so.flush(1000001);
switch (flushResult) {
    case 'pending' :
        message_txt.text = "flush is pending, waiting on user interaction.";
        break;
    case true :
        message_txt.text = "flush was successful. Requested storage space approved.";
```

```
        break;
    case false :
        message_txt.text = "flush failed. User denied request for additional
        storage.";
        break;
}
```

関連項目

[onStatus \(System.onStatus ハンドラ \)](#)

removeListener (SharedObject.removeListener メソッド)

`public static removeListener(objectName:String)`

`addListener()` メソッドを使用して追加されたリスナーを削除します。

パラメータ

`objectName:String` - 共有オブジェクトの名前を表すストリング。

戻り値

Sound

```
Object
 |
 +-Sound
```

`public class Sound`
`extends Object`

`Sound` クラスを使用すると、ムービー内のサウンドを制御することができます。ムービーの再生中にライブラリからムービークリップにサウンドを追加し、追加したサウンドを制御することができます。新しい `Sound` オブジェクトの作成時に `target` を指定しない場合は、このクラスのメソッドを使用してムービー全体のサウンドを制御できます。

`Sound` クラスのメソッドを呼び出す前に、コンストラクタ `new Sound` を使用して `Sound` オブジェクトを作成する必要があります。

プロパティ一覧

オプション	プロパティ	説明
	<code>duration:Number</code> (読み取り専用)	ミリ秒数で示したサウンドの継続時間。
	<code>id3:Object</code> (読み取り専用)	MP3 ファイルの一部であるメタデータに対するアクセスを提供します。
	<code>position:Number</code> (読み取り専用)	サウンドを再生しているミリ秒数。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

イベント一覧

イベント	説明
<code>onID3 = function()</code> {}	Sound.attachSound() または Sound.loadSound() を使用してロードした MP3 ファイルについて、新しい ID3 データが利用できるようになるたびに呼び出されます。
<code>onLoad =</code> <code>function(success: Boolean) {}</code>	サウンドがロードされると、自動的に呼び出されます。
<code>onSoundComplete =</code> <code>function() {}</code>	サウンドの再生が終了すると、自動的に呼び出されます。

コンストラクター一覧

シグネチャ	説明
<code>Sound([target:Object])</code>	指定されたムービークリップの新しい Sound オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>attachSound (id:String) : Void</code>	id パラメータで指定されたサウンドを、指定された Sound オブジェクトに割り当てます。
	<code>getBytesLoaded() : Number</code>	指定した Sound オブジェクトに対してロード（ストリーミング）されたバイト数を返します。
	<code>getBytesTotal() : Number</code>	指定した Sound オブジェクトのサイズをバイト単位で返します。
	<code>getPan() : Number</code>	最後の setPan() 呼び出しで設定されたパンレベルを返します。戻り値は -100(左)～+100(右) の整数です。
	<code>getTransform() : Object</code>	最後の Sound.setTransform() 呼び出しで設定された Sound オブジェクトのサウンド変換情報を返します。
	<code>getVolume() : Number</code>	サウンドボリュームレベルを 0～100 の整数で返します。ここで、0 はボリュームオフ、100 はフルボリュームです。
	<code>loadSound (url:String, isStreaming: Boolean) : Void</code>	MP3 ファイルを Sound オブジェクトにロードします。
	<code>setPan(value: Number) : Void</code>	左右のチャンネル（スピーカー）でサウンドを再生する方法を決定します。
	<code>setTransform (transformObject: Object) : Void</code>	Sound オブジェクトにサウンド変換情報、つまり "バランス" 情報を設定します。
	<code>setVolume(value: Number) : Void</code>	Sound オブジェクトにボリュームを設定します。
	<code>start ([secondOffset: Number], [loops:Number]) : Void</code>	最後に割り当てたサウンドの再生を開始します。再生を開始する位置は、パラメータを指定しない場合はサウンドの先頭、指定する場合は secondOffset パラメータで指定したサウンド内のポイントです。
	<code>stop([linkageID: String]) : Void</code>	idName パラメータで指定されたサウンドを停止します。パラメータを指定しないと、現在再生中のすべてのサウンドを停止します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

attachSound (Sound.attachSound メソッド)

public attachSound(id:String) : Void

id パラメータで指定されたサウンドを、指定された Sound オブジェクトに割り当てます。サウンドは、現在の SWF ファイルのライブラリ内に存在し、[リンケージプロパティ] ダイアログボックスで書き出しを指定する必要があります。サウンドの再生を開始するには、Sound.start() を呼び出します。

SWF ファイルのすべてのシーンからサウンドを制御できるようにするには、サウンドを SWF のメインタイムラインに置きます。

パラメータ

id:String - ライブラリ内の書き出されたサウンドの識別子。識別子は、[リンケージプロパティ] ダイアログボックスで設定します。

例

次の例では、my_sound にサウンド logoff_id をアタッチします。ライブラリ内のサウンドはリンクエージ識別子 logoff_id を持っています。

```
var my_sound:Sound = new Sound();  
my_sound.attachSound("logoff_id");  
my_sound.start();
```

duration (Sound.duration プロパティ)

public duration : Number (読み取り専用)

ミリ秒数で示したサウンドの継続時間。

メモ : Flash Lite 2.0 でこのプロパティがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

例

次の例では、サウンドをロードし、[出力] パネルにサウンドファイルの継続時間を表示します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    var totalSeconds:Number = this.duration/1000;
    trace(this.duration+" ms ("+Math.round(totalSeconds)+" seconds)");
    var minutes:Number = Math.floor(totalSeconds/60);
    var seconds = Math.floor(totalSeconds)%60;
    if (seconds<10) {
        seconds = "0"+seconds;
    }
    trace(minutes+":"+seconds);
};
my_sound.loadSound("song1.mp3", true);
```

次の例では、SWF ファイルにいくつかの曲をロードします。Drawing API を使用して作成したプログレスバーは、ロードの進行状況を表示します。音楽が始まってロードが完了すると、[出力] パネルに情報が表示されます。音楽が始まってロードが完了すると、ログファイルに情報が書き込まれます。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var pb_height:Number = 10;
var pb_width:Number = 100;
var pb:MovieClip = this.createEmptyMovieClip("progressBar_mc",
    this.getNextHighestDepth());
pb.createEmptyMovieClip("bar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("vBar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("stroke_mc", pb.getNextHighestDepth());
pb.createTextField("pos_txt", pb.getNextHighestDepth(), 0, pb_height, pb_width,
    22);

pb._x = 100;
pb._y = 100;

with (pb.bar_mc) {
    beginFill(0x00FF00);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
}
with (pb.vBar_mc) {
    lineStyle(1, 0x000000);
    moveTo(0, 0);
    lineTo(0, pb_height);
}
```

```

with (pb.stroke_mc) {
    lineStyle(3, 0x000000);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
}

var my_interval:Number;
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        trace("sound loaded");
    }
};
my_sound.onSoundComplete = function() {
    clearInterval(my_interval);
    trace("Cleared interval");
}
my_sound.loadSound("song3.mp3", true);
my_interval = setInterval(updateProgressBar, 100, my_sound);

function updateProgressBar(the_sound:Sound):Void {
    var pos:Number = Math.round(the_sound.position/the_sound.duration*100);
    pb.bar_mc._xscale = pos;
    pb.vBar_mc._x = pb.bar_mc._width;
    pb.pos_txt.text = pos+"%";
}

```

関連項目

[position \(Sound.position プロパティ\)](#)

getBytesLoaded (Sound.getBytesLoaded メソッド)

public getBytesLoaded():Number

指定した Sound オブジェクトに対してロード (ストリーミング) されたバイト数を返します。

getBytesLoaded() の値と getBytesTotal() の値を比較すると、サウンドの何 % がロードされたかを判断できます。

戻り値

Number - ロードされたバイト数を示す整数。

例

次の例では、SWF ファイルにロードするサウンドファイルの、ロードされたバイト数と合計バイト数を表示する 2 つのテキストフィールドをダイナミックに作成します。ファイルがロードを終了すると、テキストフィールドにはメッセージが表示されます。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
this.createTextField("message_txt", this.getNextHighestDepth(), 10,10,300,22)
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 50, 300, 40);
status_txt.autoSize = true;
status_txt.multiline = true;
status_txt.border = false;

var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        this.start();
        message_txt.text = "Finished loading";
    }
};
my_sound.onSoundComplete = function() {
    message_txt.text = "Clearing interval";
    clearInterval(my_interval);
};
my_sound.loadSound("song2.mp3", true);
var my_interval:Number;
my_interval = setInterval(checkProgress, 100, my_sound);
function checkProgress(the_sound:Sound):Void {
    var pct:Number = Math.round(the_sound.getBytesLoaded())/
        the_sound.getBytesTotal() * 100;
    var pos:Number = Math.round(the_sound.position/the_sound.duration * 100);
    status_txt.text = the_sound.getBytesLoaded()+" bytes ("+pct+"%)" +newline;
    status_txt.text += the_sound.position+" of "+the_sound.duration+" milliseconds
        ("+pos+"%)" +newline;
}
```

関連項目

[getBytesTotal \(Sound.getBytesTotal メソッド\)](#)

getBytesTotal (Sound.getBytesTotal メソッド)

public getBytesTotal() : Number

指定した Sound オブジェクトのサイズをバイト単位で返します。

戻り値

Number - 指定した Sound オブジェクトの総サイズをバイト単位で示す整数。

例

このメソッドの使用例については、Sound.getBytesLoaded() を参照してください。

関連項目

[getBytesLoaded \(Sound.getBytesLoaded メソッド\)](#)

getPan (Sound.getPan メソッド)

public getPan() : Number

最後の setPan() 呼び出しで設定されたパンレベルを返します。戻り値は -100 (左) ~ +100 (右) の整数です。0 は左右のチャンネルを等しい値に設定します。パンの設定では、SWF ファイル内の現在と今後のサウンドの左右バランスを制御します。

このメソッドの効果は setVolume() または setTransform() の効果に追加されます。

メモ : Flash Lite 2.0 でこのメソッドがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

戻り値

Number - 整数。

例

次の例では、テキストフィールドを作成し、Flash ネイティブサウンドのパンレベルの値を表示します。サウンドのリンクエージ識別子は "combo" です。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
this.createTextField("pan_txt", 1, 0, 100, 100, 100);
mix=new Sound();
mix.attachSound("combo");
mix.start();
mix.setPan(-100);
pan_txt.text = mix.getPan(this);
```

次の例を使用して、デバイスサウンドを開始することができます。Flash Lite はストリーミングサウンドをサポートしないので、再生前にサウンドをロードすることをお勧めします。

```
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success) {
    if (success) {
        my_sound.start();
    } else {
        output.text = "loading failure";
    }
};
my_sound.loadSound("song1.mp3",false);
```

関連項目

[setPan \(Sound.setPan メソッド\)](#)

getTransform (Sound.getTransform メソッド)

`public getTransform(): Object`

最後の `Sound.setTransform()` 呼び出しで設定された `Sound` オブジェクトのサウンド変換情報を返します。

メモ : Flash Lite 2.0 でこのメソッドがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

戻り値

`Object` - 指定した `Sound` オブジェクトのチャンネルパーセント値を含むプロパティを持つオブジェクト。

例

次の例では、ライブラリ内のシンボルから、4つのムービークリップを割り当てます（リンクエージェント別子 `knob_id`）。これらは、スライダ（または `knobs`）として使用され、SWF ファイルにロードされるサウンドファイルを制御します。これらのスライダは、変換オブジェクトを制御したり、サウンドファイルのバランスを調整します。詳細については、`Sound.setTransform()` を参照してください。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
var transform_obj:Object = my_sound.getTransform();

this.createEmptyMovieClip("transform_mc", this.getNextHighestDepth());
transform_mc.createTextField("transform_txt", transform_mc.getNextHighestDepth,
    0, 8, 120, 22);
transform_mc.transform_txt.html = true;
```

```

var knob_ll:MovieClip = transform_mc.attachMovie("knob_id", "ll_mc",
    transform_mc.getNextHighestDepth(), {_x:0, _y:30});
var knob_lr:MovieClip = transform_mc.attachMovie("knob_id", "lr_mc",
    transform_mc.getNextHighestDepth(), {_x:30, _y:30});
var knob_rl:MovieClip = transform_mc.attachMovie("knob_id", "rl_mc",
    transform_mc.getNextHighestDepth(), {_x:60, _y:30});
var knob_rr:MovieClip = transform_mc.attachMovie("knob_id", "rr_mc",
    transform_mc.getNextHighestDepth(), {_x:90, _y:30});

knob_ll.top = knob_ll._y;
knob_ll.bottom = knob_ll._y+100;
knob_ll.left = knob_ll._x;
knob_ll.right = knob_ll._x;
knob_ll._y = knob_ll._y+(100-transform_obj['ll']);
knob_ll.onPress = pressKnob;
knob_ll.onRelease = releaseKnob;
knob_ll.onReleaseOutside = releaseKnob;

knob_lr.top = knob_lr._y;
knob_lr.bottom = knob_lr._y+100;
knob_lr.left = knob_lr._x;
knob_lr.right = knob_lr._x;
knob_lr._y = knob_lr._y+(100-transform_obj['lr']);
knob_lr.onPress = pressKnob;
knob_lr.onRelease = releaseKnob;
knob_lr.onReleaseOutside = releaseKnob;

knob_rl.top = knob_rl._y;
knob_rl.bottom = knob_rl._y+100;
knob_rl.left = knob_rl._x;
knob_rl.right = knob_rl._x;
knob_rl._y = knob_rl._y+(100-transform_obj['rl']);
knob_rl.onPress = pressKnob;
knob_rl.onRelease = releaseKnob;
knob_rl.onReleaseOutside = releaseKnob;

knob_rr.top = knob_rr._y;
knob_rr.bottom = knob_rr._y+100;
knob_rr.left = knob_rr._x;
knob_rr.right = knob_rr._x;
knob_rr._y = knob_rr._y+(100-transform_obj['rr']);
knob_rr.onPress = pressKnob;
knob_rr.onRelease = releaseKnob;

knob_rr.onReleaseOutside = releaseKnob;

updateTransformTxt();

function pressKnob() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
}

```

```
function releaseKnob() {
    this.stopDrag();
    updateTransformTxt();
}

function updateTransformTxt() {
    var ll_num:Number = 30+100-knob_ll._y;
    var lr_num:Number = 30+100-knob_lr._y;
    var rl_num:Number = 30+100-knob_rl._y;
    var rr_num:Number = 30+100-knob_rr._y;
    my_sound.setTransform({ll:ll_num, lr:lr_num, rl:rl_num, rr:rr_num});
    transform_mc.transform_txt.htmlText = "<textformat tabStops='[0,30,60,90]'>";
    transform_mc.transform_txt.htmlText += 
        ll_num+"\t"+lr_num+"\t"+rl_num+"\t"+rr_num;
    transform_mc.transform_txt.htmlText += "</textformat>";
}
```

関連項目

[setTransform \(Sound.setTransform メソッド\)](#)

getVolume (Sound.getVolume メソッド)

public getVolume():Number

サウンドボリュームレベルを 0 ~ 100 の整数で返します。ここで、0 はボリュームオフ、100 はフルボリュームです。デフォルト設定は 100 です。

戻り値

Number - 整数。

例

次の例では、Drawing API および実行時に作成されるムービークリップを使用してスライダを作成します。ダイナミックに作成されたテキストフィールドに、SWF ファイルで再生されているサウンドの現在のボリュームレベルが表示されます。次の ActionScript を ActionScript ファイルまたは FLA ファイルに追加します。

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song3.mp3", true);

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());

knob_mc.left = knob_mc._x;
knob_mc.right = knob_mc.left+100;
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc._x = my_sound.getVolume();
```

```

with (knob_mc) {
    lineStyle(0, 0x000000);
    beginFill(0xCCCCCC);
    moveTo(0, 0);
    lineTo(4, 0);
    lineTo(4, 18);
    lineTo(0, 18);
    lineTo(0, 0);
    endFill();
}

knob_mc.createTextField("volume_txt", knob_mc.getNextHighestDepth(),
    knob_mc._width+4, 0, 30, 22);
knob_mc.volume_txt.text = my_sound.getVolume();

knob_mc.onPress = function() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
    this.isDragging = true;
};
knob_mc.onMouseMove = function() {
    if (this.isDragging) {
        this.volume_txt.text = this._x;
    }
}
knob_mc.onRelease = function() {
    this.stopDrag();
    this.isDragging = false;
    my_sound.setVolume(this._x);
};


```

関連項目

[setVolume \(Sound.setVolume メソッド\)](#)

id3 (Sound.id3 プロパティ)

public id3 : [Object](#) (読み取り専用)

MP3 ファイルの一部であるメタデータに対するアクセスを提供します。

MP3 サウンドファイルには、ファイルについてのメタデータを示す ID3 タグを含めることができます。Sound.attachSound() または Sound.loadSound() を使用してロードした MP3 サウンドに ID3 タグが含まれる場合は、これらのプロパティを調べることができます。サポートされているのは、UTF-8 文字セットを使用する ID3 タグだけです。

Flash Player 6 (6.0.40.0) 以降は、ID3 1.0 および ID3 1.1 のタグをサポートするために、`Sound.id3` プロパティを使用しています。Flash Player 7 では、ID3 2.0 (厳密には 2.3 および 2.4) のタグのサポートが追加されました。次の表に、標準の ID3 2.0 タグと、そのタグが表すコンテンツタイプを示します。これらを調べるには、`my_sound.id3.COMM`、`my_sound.id3.TIME` などの形式を使用します。MP3 ファイルには、この表に示していないタグも含めることができます。`Sound.id3` を使用すると、それらのタグにもアクセスできます。

プロパティ	説明
<code>TFLT</code>	ファイル形式
<code>TIME</code>	時刻
<code>TIT1</code>	内容の属するグループの説明
<code>TIT2</code>	タイトル / 曲名 / 内容の説明
<code>TIT3</code>	サブタイトル / 説明の追加情報
<code>TKEY</code>	最初の調
<code>TLAN</code>	言語
<code>TLEN</code>	長さ
<code>TMED</code>	メディアタイプ
<code>TOAL</code>	オリジナルのアルバム / ムービー / ショーのタイトル
<code>TOFN</code>	オリジナルのファイル名
<code>TOLY</code>	オリジナルの作詞家 / 文書作成者
<code>TOPE</code>	オリジナルのアーティスト / 演奏者
<code>TORY</code>	オリジナルのリリース年
<code>TOWN</code>	ファイルの所有者 / ライセンス保持者
<code>TPE1</code>	主な演奏者 / ソリスト
<code>TPE2</code>	バンド / オーケストラ / 伴奏
<code>TPE3</code>	指揮者 / 演奏者詳細情報
<code>TPE4</code>	翻訳、リミックス、その他の修正を行った人
<code>TPOS</code>	セット中の位置
<code>TPUB</code>	発行者
<code>TRCK</code>	トラック番号 / セット内の位置
<code>TRDA</code>	録音日

プロパティ	説明
TRSN	インターネットラジオ局の名前
TRSO	インターネットラジオ局の所有者
TSIZ	サイズ
TSRC	ISRC(国際標準録音資料コード)
TSSE	エンコードに使用したソフトウェア / ハードウェアと設定
TYER	年
WXXX	URL リンクフレーム

Flash Player 6 は、いくつかの ID31.0 タグをサポートしていました。MP3 ファイル内にこれらのタグがなく、対応する ID3 2.0 タグがある場合は、ID3 2.0 タグが ID3 1.0 のプロパティにコピーされます。次の表は、ID3 2.0 タグから ID3 1.0 プロパティへの対応を示しています。このプロセスによって、ID3 1.0 のプロパティを読み取るように作成された既存のスクリプトとの後方互換性が維持されます。

ID3 2.0 タグ	対応する ID3 1.0 プロパティ
COMM	Sound.id3.comment
TALB	Sound.id3.album
TCON	Sound.id3.genre
TIT2	Sound.id3.songname
TPE1	Sound.id3.artist
TRCK	Sound.id3.track
TYER	Sound.id3.year

例

次の例は、"song.mp3" の ID3 プロパティを追跡して [出力] パネルに表示します。

```
var my_sound:Sound = new Sound();
my_sound.onID3 = function(){
    for( var prop in my_sound.id3 ){
        trace( prop + " : " + my_sound.id3[prop] );
    }
}
my_sound.loadSound("song.mp3", false);
```

関連項目

[attachSound \(Sound.attachSound メソッド \)](#), [loadSound \(Sound.loadSound メソッド \)](#)

loadSound (Sound.loadSound メソッド)

```
public loadSound(url:String, isStreaming:Boolean) : Void
```

MP3 ファイルを Sound オブジェクトにロードします。isStreaming パラメータを使用して、サウンドがイベントサウンドかストリーミングサウンドかを指定できます。

イベントサウンドは、再生前に完全にロードされます。イベントサウンドは、ActionScript Sound クラスによって管理され、このオブジェクトのすべてのメソッドおよびプロパティに応答します。

ストリーミングサウンドは、ダウンロードと並行して再生されます。解凍処理を開始できるだけのデータを受信すると、再生が開始されます。

このメソッドでロードされたすべての MP3(イベントとストリーミングの両方)は、ユーザーのシステム上のブラウザのファイルキャッシュに保存されます。

メモ: Flash Lite 2.0 ではすべてのサウンドがイベントサウンドとして扱われるため、isStreaming パラメータを無視することができます。

パラメータ

`url:String` - サーバー上の MP3 サウンドファイルの位置。

`isStreaming:Boolean` - ブール値。サウンドがストリーミングサウンドか (true)、またはイベントサウンドか (false) を示します。

例

次の例では、イベントサウンドをロードします。イベントサウンドはロードが完了するまで再生できません。

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", false);
```

次の例では、ストリーミングサウンドをロードします。

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
```

関連項目

[onLoad \(Sound.onLoad ハンドラ\)](#)

onID3 (Sound.onID3 ハンドラ)

```
onID3 = function() {}
```

Sound.attachSound() または Sound.loadSound() を使用してロードした MP3 ファイルについて、新しい ID3 データが利用できるようになるたびに呼び出されます。このハンドラを使用すると、ポーリングなしで ID3 データにアクセスできます。1つのファイルに ID3 1.0 タグと ID3 2.0 タグの両方が存在する場合、このハンドラは 2 回呼び出されます。

例

次の例では、DataGrid コンポーネントのインスタンスに song1.mp3 の ID3 プロパティを表示します。id3_dg というインスタンス名を持つ DataGrid をドキュメントに追加し、次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
import mx.controls.gridclasses.DataGridColumn;
var id3_dg:mx.controls.DataGrid;
id3_dg.move(0, 0);
id3_dg.setSize(Stage.width, Stage.height);
var property_dgc:DataGridColumn = id3_dg.addColumn(new
    DataGridColumn("property"));
property_dgc.width = 100;
property_dgc.headerText = "ID3 Property";
var value_dgc:DataGridColumn = id3_dg.addColumn(new DataGridColumn("value"));
value_dgc.width = id3_dg._width-property_dgc.width;
value_dgc.headerText = "ID3 Value";

var my_sound:Sound = new Sound();
my_sound.onID3 = function() {
trace("onID3 called at "+getTimer()+" ms.");
for (var prop in this.id3) {
id3_dg.addItem({property:prop, value:this.id3[prop]});
}
};
my_sound.loadSound("song1.mp3", true);
```

関連項目

[attachSound \(Sound.attachSound メソッド\)](#), [id3 \(Sound.id3 プロパティ\)](#), [loadSound \(Sound.loadSound メソッド\)](#)

onLoad (Sound.onLoad ハンドラ)

onLoad = function(success:Boolean) {}

サウンドがロードされると、自動的に呼び出されます。このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。匿名関数または名前付き関数を使用できます。それぞれの例については、[Sound.onSoundComplete](#) を参照してください。このハンドラは、[mySound.loadSound\(\)](#) を呼び出す前に定義する必要があります。

パラメータ

success:Boolean - ブール値。my_sound のロードが成功した場合は true、それ以外の場合は false を返します。

例

次の例では、新しい Sound オブジェクトを作成して、サウンドをロードします。サウンドのロードは onLoad ハンドラにより処理されます。音楽が正常にロードされた後に再生を開始できます。新しい FLA ファイルを作成し、次の ActionScript を FLA ファイルまたは AS ファイルに追加します。このコード例が動作するには、song1.mp3 という名前の MP3 ファイルを、FLA ファイルまたは AS ファイルと同じディレクトリに置く必要があります。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// If the sound loads, play it; if not, trace failure loading.
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        my_sound.start();
        status_txt.text = "Sound loaded";
    } else {
        status_txt.text = "Sound failed";
    }
};
// Load the sound.
my_sound.loadSound("song1.mp3", true);
```

関連項目

[loadSound \(Sound.loadSound メソッド\)](#)

onSoundComplete (Sound.onSoundComplete ハンドラ)

```
onSoundComplete = function() {}
```

サウンドの再生が終了すると、自動的に呼び出されます。このハンドラを使用して、サウンドの再生が終了したときに SWF ファイル内のイベントをトリガすることができます。

このイベントハンドラが呼び出されたときに実行される関数を定義する必要があります。匿名関数でも名前付き関数でも使用できます。

例

シンタックス 1: 次の例では、匿名関数を使用します。

```
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = function() {
    trace("mySoundID completed");
};
my_sound.start();
```

シンタックス 2: 次の例では、名前付き関数を使用します。

```
function callback1() {  
    trace("mySoundID completed");  
}  
var my_sound:Sound = new Sound();  
my_sound.attachSound("mySoundID");  
my_sound.onSoundComplete = callback1;  
my_sound.start();
```

関連項目

[onLoad \(Sound.onload ハンドラ\)](#)

position (Sound.position プロパティ)

public position : [Number](#) (読み取り専用)

サウンドを再生しているミリ秒数。サウンドをループしている場合、サウンド位置は各ループの最初に 0 にリセットされます。

メモ: Flash Lite 2.0 でこのプロパティがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

例

このプロパティの使用例については [Sound.duration](#) を参照してください。

関連項目

[duration \(Sound.duration プロパティ\)](#)

setPan (Sound.setPan メソッド)

public setPan(value:[Number](#)) : [Void](#)

左右のチャンネル(スピーカー)でサウンドを再生する方法を決定します。モノラルサウンドの場合、*pan* は左右いずれのスピーカーからサウンドを再生するかを決定します。

メモ: Flash Lite 2.0 でこのメソッドがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

パラメータ

value: [Number](#) - サウンドの左右バランスを指定する整数。有効値の範囲は -100 ~ 100 です。ここで、-100 は左のチャンネルのみを使用し、100 は右のチャンネルのみを使用します。0 は 2 つのチャンネル間でサウンドのバランスを等しくします。

例

このメソッドの使用例については、`Sound.getPan()` を参照してください。

関連項目

`attachSound` (`Sound.attachSound` メソッド), `getPan` (`Sound.getPan` メソッド),
`setTransform` (`Sound.setTransform` メソッド), `setVolume` (`Sound.setVolume` メソッド), `start` (`Sound.start` メソッド)

setTransform (Sound.setTransform メソッド)

`public setTransform(transformObject:Object) : Void`

`Sound` オブジェクトにサウンド変換情報、つまり "バランス" 情報を設定します。

`soundTransformObject` パラメータは、汎用 `Object` オブジェクトのコンストラクタメソッドを使用して作成するオブジェクトです。このオブジェクトには、左右のチャンネル(スピーカー)間でサウンドをどのように配分するかを指定するパラメータがあります。

サウンドは、大量のディスク領域およびメモリを必要とします。ステレオサウンドはモノラルサウンドの2倍のデータを使用するため、一般には 22 kHz、6 ビットのモノラルサウンドを使用するのが最善です。`setTransform()` メソッドを使用すると、モノラルサウンドをステレオサウンドとして再生したり、ステレオサウンドをモノラルサウンドのように再生したり、またサウンドに特殊効果を追加することができます。

メモ: Flash Lite 2.0 でこのメソッドがサポートされるのは、Flash ネイティブサウンドのみです。ホストデバイスに固有のサウンド形式はサポートされません。

`soundTransformObject` のプロパティは、次のとおりです。

ll - 左スピーカーで再生する左入力データの量を指定するパーセント値 (0 ~ 100)

lr - 左スピーカーで再生する右入力データの量を指定するパーセント値 (0 ~ 100)

rr - 右スピーカーで再生する右入力データの量を指定するパーセント値 (0 ~ 100)

rl - 右スピーカーで再生する左入力データの量を指定するパーセント値 (0 ~ 100)

パラメータの最終結果は、次の式で表されます。

```
leftOutput = left_input ~ ll + right_input ~ lr  
rightOutput = right_input ~ rr + left_input ~ rl
```

`left_input` または `right_input` の値は、SWF ファイル内のサウンドのタイプ(ステレオまたはモノラル)によって決定されます。

ステレオサウンドのデフォルトでは、左スピーカーと右スピーカーの間でサウンド入力を等しく分割するように、次の変換設定になっています。

```
ll = 100  
lr = 0
```

```
rr = 100  
rl = 0
```

モノラルサウンドのデフォルトでは、左スピーカーですべてのサウンド入力を再生するように、次の変換設定になっています。

```
ll = 100  
lr = 100  
rr = 0  
rl = 0
```

パラメータ

`transformObject:Object` - 汎用の Object クラス用コンストラクタで作成したオブジェクト。

例

次の例では、`setTransform()` メソッドでは実現でき、`setVolume()` メソッドや `setPan()` メソッドでは両方を組み合っても実現できない設定を示します。

次のコードでは、新しい `soundTransformObject` オブジェクトを作成し、両方のチャンネルのサウンドが左チャンネルでのみ再生されるように、オブジェクトのプロパティを設定します。

```
var mySoundTransformObject:Object = new Object();  
mySoundTransformObject.ll = 100;  
mySoundTransformObject.lr = 100;  
mySoundTransformObject.rr = 0;  
mySoundTransformObject.rl = 0;
```

`soundTransformObject` オブジェクトを `Sound` オブジェクトに適用するには、次のように `setTransform()` メソッドを使用してオブジェクトを `Sound` オブジェクトに渡します。

```
my_sound.setTransform(mySoundTransformObject);
```

次の例では、ステレオサウンドをモノラルとして再生します。`soundTransformObjectMono` オブジェクトは以下のパラメータを使用します。

```
var mySoundTransformObjectMono:Object = new Object();  
mySoundTransformObjectMono.ll = 50;  
mySoundTransformObjectMono.lr = 50;  
mySoundTransformObjectMono.rr = 50;  
mySoundTransformObjectMono.rl = 50;  
my_sound.setTransform(mySoundTransformObjectMono);
```

次の例では、半分のボリュームで左チャンネルを再生し、右チャンネルに左チャンネルの残りの部分を追加します。`soundTransformObjectHalf` オブジェクトは以下のパラメータを使用します。

```
var mySoundTransformObjectHalf:Object = new Object();  
mySoundTransformObjectHalf.ll = 50;  
mySoundTransformObjectHalf.lr = 0;  
mySoundTransformObjectHalf.rr = 100;  
mySoundTransformObjectHalf.rl = 50;  
my_sound.setTransform(mySoundTransformObjectHalf);  
var mySoundTransformObjectHalf:Object = {ll:50, lr:0, rr:100, rl:50};
```

関連項目

[Object.getTransform \(Sound.getTransform メソッド\)](#)

setVolume (Sound.setVolume メソッド)

public setVolume(value:Number) : Void

Sound オブジェクトにボリュームを設定します。

パラメータ

value:Number - ボリュームレベルを表す 0 ~ 100 の数値。100 はフルボリュームであり、0 はボリュームなしです。デフォルト設定は 100 です。

例

このメソッドの使用例については、[Sound.getVolume\(\)](#) を参照してください。

関連項目

[setPan \(Sound.setPan メソッド\)](#), [setTransform \(Sound.setTransform メソッド\)](#)

Sound コンストラクタ

public Sound([target:Object])

指定されたムービークリップの新しい Sound オブジェクトを作成します。ターゲットインスタンスを指定しないと、Sound オブジェクトはムービー内のすべてのサウンドを制御します。

パラメータ

target:Object (オプション) - Sound オブジェクトを適用するムービークリップインスタンス。

例

次の例では、global_sound という新しい Sound オブジェクトを作成します。2 行目で、
setVolume() メソッドを呼び出してムービー内のすべてのサウンドのボリュームを 50% に調整します。

```
var global_sound:Sound = new Sound();
global_sound.setVolume(50);
```

次の例では、新しい Sound オブジェクトを作成し、それにターゲットムービークリップ my_mc を渡し、start メソッドを呼び出して my_mc 内のサウンドを開始します。

```
var movie_sound:Sound = new Sound(my_mc);
movie_sound.start();
```

start (Sound.start メソッド)

```
public start([secondOffset:Number], [loops:Number]) : Void
```

最後に割り当てたサウンドの再生を開始します。再生を開始する位置は、パラメータを指定しない場合はサウンドの先頭、指定する場合は secondOffset パラメータで指定したサウンド内のポイントです。

パラメータ

secondOffset:Number (オプション) - 特定のポイントでサウンド再生を開始するためのパラメータ。たとえば、30 秒のサウンドがあり、ちょうど中間からサウンドの再生を開始するには、

secondOffset パラメータに 15 を指定します。サウンドは再生開始が 15 秒遅延するのではなく、15 秒の時点から開始されます。

loops:Number (オプション) - サウンドを連続再生する回数を指定するためのパラメータ。サウンドがストリーミングサウンドの場合、このパラメータは使用できません。

例

次の例では、新しい Sound オブジェクトを作成して、サウンドをロードします。onLoad ハンドラによりサウンドがロードされ、音楽が正常にロードされた後に再生を開始できます。start() メソッドを使用して、サウンドの再生が開始します。新しい FLA ファイルを作成し、次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。このコード例が動作するには、"song1.mp3" という名前の MP3 ファイルを、FLA ファイルまたは AS ファイルと同じディレクトリに置く必要があります。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// If the sound loads, play it; if not, trace failure loading.
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        my_sound.start();
        status_txt.text = "Sound loaded";
    } else {
        status_txt.text = "Sound failed";
    }
};
// Load the sound.
my_sound.loadSound("song1.mp3", true);
```

関連項目

[stop \(Sound.stop メソッド \)](#)

stop (Sound.stop メソッド)

```
public stop([linkageID:String]) : Void
```

idName パラメータで指定されたサウンドを停止します。パラメータを指定しないと、現在再生中のすべてのサウンドを停止します。

パラメータ

linkageID:String (オプション) - 再生を停止する特定のサウンドを指定するパラメータ。idName パラメータは、引用符 (" ") で囲む必要があります。

例

次の例では、stop_btn および play_btn の 2 つのボタンを使用し、SWF ファイルにロードされるサウンドの再生を制御します。ドキュメントに 2 つのボタンを追加し、次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);

stop_btn.onRelease = function() {
    trace("sound stopped");
    my_sound.stop();
};

play_btn.onRelease = function() {
    trace("sound started");
    my_sound.start();
};
```

関連項目

[start \(Sound.start メソッド\)](#)

Stage

```
Object
```

```
|
```

```
+--Stage
```

```
public class Stage
extends Object
```

Stage クラスはトップレベルのクラスで、コンストラクタを実行しなくともそのメソッド、プロパティ、およびハンドラを使用できます。このクラスのメソッドとプロパティを使用して、SWF ファイルの境界に関する情報にアクセスし、操作できます。

プロパティ一覧

オプション	プロパティ	説明
static	<code>align:String</code>	Flash Player またはブラウザ内での SWF ファイルの整列設定を示します。
static	<code>height:Number</code>	読み取り専用プロパティ。ステージの現在の高さをピクセル単位で示します。
static	<code>scaleMode:String</code>	Flash Player 内での SWF ファイルの拡大 / 縮小に関する現在の設定を示します。
static	<code>width:Number</code>	読み取り専用プロパティ。ステージの現在の幅をピクセル単位で示します。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
<code>onResize = function() {}</code>	Stage.scaleMode が noScale に設定されているとき、SWF ファイルのサイズが変更されると呼び出されます。

メソッド一覧

オプション	シグネチャ	説明
static	<code>addListener(listener:Object) : Void</code>	Stage.scaleMode = "noScale" である場合に、SWF ファイルのサイズ変更を検出します。
static	<code>removeListener(listener:Object) : Boolean</code>	addListener() で作成したオブジェクトを削除します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (Stage.addListener メソッド)

```
public static addListener(listener:Object) : Void
```

Stage.scaleMode = "noScale" である場合に、SWF ファイルのサイズ変更を検出します。addListener() メソッドは、ムービークリップのデフォルトの拡大 / 縮小設定(showAll) や、その他の拡大 / 縮小設定(exactFit および noBorder) とは併用できません。

addListener() を使用するには、まずリスナーオブジェクトを作成する必要があります。Stage オブジェクトのリスナーオブジェクトは、Stage.onResize から通知を受け取ります。

パラメータ

listener:Object - Stage.onResize イベントからのコールバック通知を待機するオブジェクト。

例

次の例では、新しいリスナーオブジェクト stageListener を作成します。次に、stageListener を使用して onResize を呼び出し、さらに onResize の起動に応じて呼び出す関数を定義します。最後に、stageListener オブジェクトを Stage オブジェクトのコールバックリストに追加します。リスナーオブジェクトを使用すると、複数のオブジェクトがサイズ変更通知を取得できます。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,  
    22);  
var stageListener:Object = new Object();  
stageListener.onResize = function() {  
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;  
};  
Stage.scaleMode = "noScale";  
Stage.addListener(stageListener);
```

関連項目

[onResize \(Stage.onResize イベントリスナー\)](#),
[removeListener \(Stage.removeListener メソッド\)](#)

align (Stage.align プロパティ)

public static align : String

Flash Player またはブラウザ内での SWF ファイルの整列設定を示します。

次の表は、align プロパティの値の一覧です。表にない値を使用すると、SWF ファイルはデフォルトの位置である Flash Player またはブラウザ領域の中央に配置されます。

値	垂直方向	水平方向
"T"	上	中央
"B"	下	中央
"L"	中央	左
"R"	中央	右
"TL"	上	左
"TR"	上	右
"BL"	下	左
"BR"	下	右

例

次の例では、SWF ファイルの異なる整列を示します。ComboBox インスタンスを stageAlign_cb のインスタンス名を持つドキュメントに追加します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var stageAlign_cb:mx.controls.ComboBox;
stageAlign_cb.dataProvider = ['T', 'B', 'L', 'R', 'TL', 'TR', 'BL', 'BR'];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var align:String = evt.target.selectedItem;
    Stage.align = align;
};
stageAlign_cb.addEventListener("change", cbListener);
Stage.scaleMode = "noScale";
ComboBox から異なる整列設定を選択します。
```

height (Stage.height プロパティ)

```
public static height : Number
```

読み取り専用プロパティ。ステージの現在の高さをピクセル単位で示します。Stage.scaleMode の値が noScale である場合、height プロパティは Flash Player の高さを表します。Stage.scaleMode の値が noScale 以外の場合、height プロパティは SWF ファイルの高さを表します。

例

次の例では、新しいリスナーオブジェクト stageListener を作成します。次に、myListener を使用して onResize を呼び出し、さらに onResize の起動に応じて呼び出す関数を定義します。最後に、myListener オブジェクトを Stage オブジェクトのコールバックリストに追加します。リスナーオブジェクトを使用すると、複数のオブジェクトがサイズ変更通知を取得できます。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,  
    22);  
var stageListener:Object = new Object();  
stageListener.onResize = function() {  
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;  
};  
Stage.scaleMode = "noScale";  
Stage.addListener(stageListener);
```

関連項目

[align \(Stage.align プロパティ\)](#), [scaleMode \(Stage.scaleMode プロパティ\)](#),
[width \(Stage.width プロパティ\)](#)

onResize (Stage.onResize イベントリスナー)

```
onResize = function() {}
```

Stage.scaleMode が noScale に設定されているとき、SWF ファイルのサイズが変更されると呼び出されます。このイベントハンドラを使用して、SWF ファイルのサイズ変更に応じてステージ上のオブジェクトを配置する関数を記述できます。

例

次の例では、ステージのサイズが変更されたときに、[出力] パネルにメッセージを表示します。

```
Stage.scaleMode = "noScale"  
var myListener:Object = new Object();  
myListener.onResize = function () {  
    trace("Stage size is now " + Stage.width + " by " + Stage.height);  
}  
Stage.addListener(myListener);  
// later, call Stage.removeListener(myListener)
```

関連項目

[scaleMode \(Stage.scaleMode プロパティ\)](#), [addListener \(Stage.addListener メソッド\)](#), [removeListener \(Stage.removeListener メソッド\)](#)

removeListener (Stage.removeListener メソッド)

public static removeListener(listener:[Object](#)) : Boolean

addListener() で作成したオブジェクトを削除します。

パラメータ

listener:[Object](#) - addListener() を使用してオブジェクトのコールバックに追加されたオブジェクト。

戻り値

[Boolean](#) - ブール値。

例

次の例では、ダイナミックに作成されたテキストフィールドに、ステージのサイズを表示します。ステージのサイズを変更すると、テキストフィールドの値も更新されます。remove_btn というインスタンス名のボタンを作成します。タイムラインのフレーム 1 に次の ActionScript を追加します。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,  
    22);  
stageSize_txt.autoSize = true;  
stageSize_txt.border = true;  
var stageListener:Object = new Object();  
stageListener.onResize = function() {  
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;  
};  
Stage.addListener(stageListener);  
  
remove_btn.onRelease = function() {  
    stageSize_txt.text = "Removing Stage listener...";  
    Stage.removeListener(stageListener);  
}
```

[制御]-[ムービープレビュー]を選択してこの例をテストします。テキストフィールドに表示される値は、テスト環境のサイズを変更すると更新されます。remove_btn をクリックすると、リスナーが削除され、テキストフィールドの値は更新されなくなります。

関連項目

[addListener \(Stage.addListener メソッド\)](#)

scaleMode (Stage.scaleMode プロパティ)

```
public static scaleMode : String
```

Flash Player 内での SWF ファイルの拡大 / 縮小に関する現在の設定を示します。scaleMode プロパティは、SWF ファイルに特定の拡大 / 縮小モードを適用します。デフォルトでは、[パブリッシュ設定] ダイアログボックスに設定されている HTML パラメータが SWF ファイルに適用されます。

scaleMode プロパティには、"exactFit"、"showAll"、"noBorder"、"noScale" の各値を使用することができます。他の値を使用すると、scaleMode プロパティはデフォルトの "showAll" に設定されます。

- デフォルトの showAll を指定すると、指定された領域内に Flash コンテンツ全体が元の縦横比を維持したまま、歪まずに表示されます。ただし、アプリケーションの両側に境界枠が表示されることがあります。
- noBorder を指定すると、指定された領域いっぱいに Flash コンテンツがサイズ調整されて、歪まずに表示されます。ただし、アプリケーションの元の縦横比を保つために、ある程度トリミングされることがあります。
- exactFit を指定すると、指定された領域にちょうど収まるように Flash コンテンツ全体が表示されますが、元の縦横比は保たれません。ゆがみが発生する場合もあります。
- noScale を指定すると、Flash コンテンツのサイズが固定され、プレーヤーウィンドウのサイズが変更された場合でも、サイズが維持されます。プレーヤーウィンドウが Flash コンテンツよりも小さい場合は、トリミングされることがあります。

メモ : デフォルト設定は showAll ですが、ムービープレビュー モードでは、noScale がデフォルト設定になります。

例

次の例では、SWF ファイルのさまざまな拡大 / 縮小設定を示します。ComboBox インスタンスを scaleMode_cb のインスタンス名を持つドキュメントに追加します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var scaleMode_cb:mx.controls.ComboBox;
scaleMode_cbdataProvider = ["showAll", "exactFit", "noBorder", "noScale"];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var scaleMode_str:String = evt.target.selectedItem;
    Stage.scaleMode = scaleMode_str;
};
scaleMode_cb.addEventListener("change", cbListener);
```

別の例については、www.adobe.com/go/learn_flt_samples_and_tutorials_jp の "ActionScript" サンプル フォルダにある "stagesize.fla" ファイルを参照してください。ご使用の Flash Lite バージョンの "Samples_and_Tutorials.zip" ファイルをダウンロードして解凍し、ActionScript フォルダに移動してサンプル ファイルにアクセスします。

width (Stage.width プロパティ)

```
public static width : Number
```

読み取り専用プロパティ。ステージの現在の幅をピクセル単位で示します。Stage.scaleMode の値が "noScale" である場合、width プロパティは Flash Player の幅を表します。つまり、Flash Player のウィンドウのサイズを変更すると、それに合わせて Stage.width が変化します。Stage.scaleMode の値が "noScale" 以外の場合、width はオーサリング時に [ドキュメントプロパティ] ダイアログボックスで設定した SWF ファイルの幅を表します。つまり、Flash Player のウィンドウのサイズを変更しても width の値は一定になります。

例

次の例では、新しいリスナーオブジェクト stageListener を作成します。次に、stageListener を使用して onResize を呼び出し、さらに onResize の起動に応じて呼び出す関数を定義します。最後に、stageListener オブジェクトを Stage オブジェクトのコールバックリストに追加します。リスナーオブジェクトを使用すると、複数のオブジェクトがサイズ変更通知を取得できます。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,  
    22);  
var stageListener:Object = new Object();  
stageListener.onResize = function() {  
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;  
};  
Stage.scaleMode = "noScale";  
Stage.addListener(stageListener);
```

関連項目

[align \(Stage.align プロパティ\)](#), [height \(Stage.height プロパティ\)](#),
[scaleMode \(Stage.scaleMode プロパティ\)](#)

String

```
Object  
|  
+-String
```

```
public class String  
extends Object
```

String クラスは、ストリングプリミティブデータ型のラッパーです。このメソッドとプロパティを使用して、プリミティブストリング値の型を操作できます。String() 関数を使用して、任意のオブジェクトの値をストリングに変換できます。

`concat()`、`fromCharCode()`、`slice()`、`substr()`を除く `String` クラスのすべてのメソッドは汎用メソッドです。つまり、メソッドが `toString()` を呼び出した後で、メソッドの操作が実行されます。これらのメソッドは `String` オブジェクト以外のオブジェクトでも使用できます。

すべてのストリングインデックスはゼロから始まるため、各ストリング `x` の最終文字のインデックスは `x.length - 1` のようになります。

`String` クラスのメソッドを呼び出すには、コンストラクタメソッド `new String` を使用するか、ストリングリテラル値を使用します。ストリングリテラルを指定すると、ActionScript インタプリタはそれをテンポラリ `String` オブジェクトに自動変換し、その後、テンポラリ `String` オブジェクトを破棄します。ストリングリテラルで `String.length` プロパティを使用することもできます。

ストリングリテラルと `String` オブジェクトを混同しないように注意してください。次の例では、コードの1行目でストリングリテラル `first_string` を作成し、2行目で `String` オブジェクト `second_string` を作成します。

```
var first_string:String = "foo"  
var second_string:String = new String("foo")
```

`String` オブジェクトを特に使用する必要がない限り、ストリングリテラルを使用してください。

プロパティ一覧

オプション	プロパティ	説明
	<code>length:Number</code>	指定した <code>String</code> オブジェクト内にある文字数を表す整数です。

`Object` クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>String (value:String)</code>	新しい <code>String</code> オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>charAt(index: Number) : String</code>	パラメータ index で指定された位置にある文字を返します。
	<code>charCodeAt(index: Number) : Number</code>	index で指定された文字を表す 0 ~ 65535 の 16 ビット整数を返します。
	<code>concat(value:Object) : String</code>	String オブジェクトの値とパラメータを連結し、新しく形成したストリングを返します。元の値 my_str は変更されません。
static	<code>fromCharCode() : String</code>	パラメータ内の Unicode 値に対応する文字をストリングとして返します。
	<code>indexOf(value:String, [startIndex:Number]) : Number</code>	ストリング内を検索し、ストリング内の startIndex 以降の位置で見つかった最初の value の位置を返します。
	<code>lastIndexOf(value:String, [startIndex:Number]) : Number</code>	ストリングを右から左へと探し、ストリング内で startIndex の前に見つかった最後の value のインデックスを返します。
	<code>slice(start:Number, end:Number) : String</code>	返されるストリングには、start 文字から end 文字の前までのすべての文字が含まれます。
	<code>split(delimiter: String, [limit: Number]) : Array</code>	指定された delimiter パラメータがある各位置で String オブジェクトをサブストリングに分割し、そのサブストリングを配列として返します。
	<code>substr(start:Number, length:Number) : String</code>	ストリング内で start パラメータで指定されたインデックスから length パラメータで指定された文字数までの文字を返します。
	<code>substring(start: Number, end: Number) : String</code>	start パラメータと end パラメータで指定された点の間の文字をストリングとして返します。
	<code>toLowerCase() : String</code>	String オブジェクトのコピーを返します。すべての大文字が小文字に変換されます。
	<code>toString() : String</code>	プロパティがストリングかどうかに関係なく、オブジェクトのプロパティをストリングとして返します。
	<code>toUpperCase() : String</code>	String オブジェクトのコピーを返します。すべての小文字が大文字に変換されます。
	<code>valueOf() : String</code>	ストリングを返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

charAt (String.charAt メソッド)

public charAt(index:Number) : String

パラメータ index で指定された位置にある文字を返します。index に指定した値が 0 ~ (string.length - 1) の範囲内でない場合は、空のストリングを返します。

このメソッドは String.charCodeAt() に似ていますが、16 ビット整数の文字コードではなく文字が返される点が異なります。

パラメータ

index:Number - スtring 内の文字の位置を指定する整数。最初の文字の位置は 0 で、最後の文字の位置は my_str.length-1 です。

戻り値

String - 指定されたインデックス位置の文字。指定されたインデックスがこの String のインデックスの範囲外である場合は空の String が返されます。

例

次の例では、ストリング "Chris" の最初の文字に対してこのメソッドを呼び出します。

```
var my_str:String = "Chris";  
var firstChar_str:String = my_str.charAt(0);  
trace(firstChar_str); // output: C
```

関連項目

[charCodeAt \(String.charCodeAt メソッド \)](#)

charCodeAt (String.charCodeAt メソッド)

public `charCodeAt(index:Number) : Number`

`index` で指定された文字を表す 0 ~ 65535 の 16 ビット整数を返します。`index` に指定した値が 0 ~ (`string.length - 1`) の範囲内ではない場合は、`NaN` を返します。

このメソッドは、`String.charAt()` と似ていますが、文字ではなく 16 ビット整数文字コードを返す点が異なります。

パラメータ

`index:Number` - ストリング内の文字の位置を指定する整数。最初の文字の位置は 0 で、最後の文字の位置は `my_str.length - 1` です。

戻り値

`Number` - `index` で指定された文字を表す整数。

例

次の例では、ストリング "Chris" の最初の文字に対してこのメソッドを呼び出します。

```
var my_str:String = "Chris";
var firstChar_num:Number = my_str.charCodeAt(0);
trace(firstChar_num); // output: 67
```

関連項目

[charAt \(String.charAt メソッド \)](#)

concat (String.concat メソッド)

public `concat(value:Object) : String`

`String` オブジェクトの値とパラメータを連結し、新しく形成したストリングを返します。元の値 `my_str` は変更されません。

パラメータ

`value:Object` - `value1[...valueN]` 連結される値。

戻り値

`String` - ストリング。

例

次の例では、ストリングを2つ作成し、String.concat()を使用してそれを連結します。

```
var stringA:String = "Hello";
var stringB:String = "World";
var combinedAB:String = stringA.concat(" ", stringB);
trace(combinedAB); // output: Hello World
```

fromCharCode (String.fromCharCode メソッド)

public static fromCharCode(): String

パラメータ内の Unicode 値に対応する文字をストリングとして返します。

戻り値

`String` - 指定された Unicode 文字コードのストリング値。

例

次の例では、fromCharCode() メソッドを使用して、電子メールアドレスに @ 文字を挿入します。

```
var address_str:String = "dog"+String.fromCharCode(64)+"house.net";
trace(address_str); // output: dog@house.net
```

indexOf (String.indexOf メソッド)

public indexOf(value:String, [startIndex:Number]): Number

ストリング内を検索し、ストリング内の startIndex 以降の位置で見つかった最初の value の位置を返します。このインデックスはゼロから始まります。つまりストリングの最初の文字は、インデックス 1ではなくインデックス 0 にあると見なされます。value が見つからない場合、メソッドは -1 を返します。

パラメータ

`value:String` - ストリング。検索対象のサブストリングです。

`startIndex:Number` (オプション) - 検索の開始インデックスを指定する整数

戻り値

`Number` - 最初に見つかった指定のサブストリングの位置。見つからなかった場合は -1 を返します。

例

次の例では、`indexOf()`を使用して文字とサブストリングのインデックスを返します。

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.indexOf("L");
trace(index); // output: 0

index = searchString.indexOf("l");
trace(index); // output: 14

index = searchString.indexOf("i");
trace(index); // output: 6

index = searchString.indexOf("ipsum");
trace(index); // output: 6

index = searchString.indexOf("i", 7);
trace(index); // output: 19

index = searchString.indexOf("z");
trace(index); // output: -1
```

関連項目

[lastIndexOf \(String.lastIndexOf メソッド\)](#)

lastIndexOf (String.lastIndexOf メソッド)

`public lastIndexOf(value:String, [startIndex:Number]) : Number`

ストリングを右から左へと探し、ストリング内で `startIndex` の前に見つかった最後の `value` のインデックスを返します。このインデックスはゼロから始まります。つまりストリングの最初の文字は、インデックス 1ではなくインデックス 0 にあると見なされます。`value` が見つからない場合、メソッドは -1 を返します。

パラメータ

`value:String` - 検査対象のストリング。

`startIndex:Number` (オプション) - `value` を検索する開始位置を指定する整数。

戻り値

`Number` - 最後に見つかった指定のサブストリングの位置。見つからなかった場合は -1 を返します。

例

次の例では、`lastIndexOf()` を使用して特定の文字のインデックスを返す方法を示します。

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.lastIndexOf("L");
trace(index); // output: 0

index = searchString.lastIndexOf("l");
trace(index); // output: 14

index = searchString.lastIndexOf("i");
trace(index); // output: 19

index = searchString.lastIndexOf("ipsum");
trace(index); // output: 6

index = searchString.lastIndexOf("i", 18);
trace(index); // output: 6

index = searchString.lastIndexOf("z");
trace(index); // output: -1
```

関連項目

[indexOf \(String.indexOf メソッド\)](#)

length (String.length プロパティ)

`public length : Number`

指定した `String` オブジェクト内にある文字数を表す整数です。

すべてのストリングインデックスはゼロから始まるため、各ストリング `x` の最終文字のインデックスは `x.length - 1` のようになります。

例

次の例では、新しい `String` オブジェクトを作成し、`String.length` を使用して文字数をカウントします。

```
var my_str:String = "Hello world!";
trace(my_str.length); // output: 12
```

次の例では、`0` から `my_str.length` までループします。このコードでは、ストリング内の文字をチェックします。ストリングに@文字が含まれる場合には、[出力] パネルに `true` が表示されます。`@` 文字がない場合には、[出力] パネルに `false` が表示されます。

```
function checkAtSymbol(my_str:String):Boolean {
    for (var i = 0; i<my_str.length; i++) {
```

```

if (my_str.charAt(i) == "@") {
    return true;
}
}
return false;
}

trace(checkAtSymbol("dog@house.net")); // output: true
trace(checkAtSymbol("Chris")); // output: false

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

```

slice (String.slice メソッド)

`public slice(start:Number, end:Number) : String`

返されるストリングには、start 文字から end 文字の前までのすべての文字が含まれます。元の String オブジェクトは変更されません。end パラメータを指定しないと、サブストリングの終わりはストリングの終わりです。start で指定されたインデックスの文字が、end で指定されたインデックス文字と同じか、その右側にある場合、メソッドは空の文字列を返します。

パラメータ

`start:Number` - スライスの始点のゼロから始まるインデックス。start が負の数値の場合、始点はストリングの終わりから決定されます。このとき、-1が最後の文字になります。

`end:Number` - スライスの終点のインデックスより大きい整数値。end パラメータで指定されたインデックス位置の文字は、取り出されるストリングには含まれません。このパラメータを省略すると、`String.length` が使用されます。end が負の数値の場合、終点はストリングの終わりからカウントされて決定されます。このとき、-1が最後の文字になります。

戻り値

`String` - 指定されたストリングのサブストリング。

例

次の例では、変数 `my_str` を作成し、それに `String` 値を割り当てた後、`start` パラメータと `end` パラメータにさまざまな値を使用して `slice()` メソッドを呼び出します。`slice()` の各呼び出しは、[出力] パネルに出力を表示する `trace()` ステートメントでラップされています。

```

// Index values for the string literal
// positive index: 0 1 2 3 4
// string: L o r e m
// negative index: -5 -4 -3 -2 -1

```

```
var my_str:String = "Lorem";  
  
// slice the first character  
trace("slice(0,1): "+my_str.slice(0, 1)); // output: slice(0,1): L  
trace("slice(-5,1): "+my_str.slice(-5, 1)); // output: slice(-5,1): L  
  
// slice the middle three characters  
trace("slice(1,4): "+my_str.slice(1, 4)); // slice(1,4): ore  
trace("slice(1,-1): "+my_str.slice(1, -1)); // slice(1,-1): ore  
  
// slices that return empty strings because start is not to the left of end  
trace("slice(1,1): "+my_str.slice(1, 1)); // slice(1,1):  
trace("slice(3,2): "+my_str.slice(3, 2)); // slice(3,2):  
trace("slice(-2,2): "+my_str.slice(-2, 2)); // slice(-2,2):  
  
// slices that omit the end parameter use String.length, which equals 5  
trace("slice(0): "+my_str.slice(0)); // slice(0): Lorem  
trace("slice(3): "+my_str.slice(3)); // slice(3): em  
  
www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。
```

関連項目

[substr \(String.substr メソッド\)](#), [substring \(String.substring メソッド\)](#)

split (String.split メソッド)

public split(delimiter:String, [limit:Number]) : Array

指定された delimiter パラメータがある各位置で String オブジェクトをサブストリングに分割し、そのサブストリングを配列として返します。区切り記号として空のストリング("")を使用すると、ストリング内の各文字がエレメントとして配列に挿入されます。

delimiter パラメータが未定義の場合は、ストリング全体が返される配列の最初のエレメントに挿入されます。

パラメータ

delimiter:String - ストリング。my_str を分割する文字またはストリングです。

limit:Number (オプション) - 配列に挿入する項目数。

戻り値

Array - my_str のサブストリングを含む配列。

例

次の例では、5つのエレメントで構成される配列を返します。

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
// output:
P
A
T
S
Y
```

次の例では、2つのエレメント "P" および "A" で構成される配列を返します。

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(", ", 2);
trace(my_array); // output: P,A
```

次の例に示すように、delimiter パラメータに空のストリング ("") を使用すると、ストリング内の各文字がエレメントとして配列に挿入されます。

```
var my_str:String = new String("Joe");
var my_array:Array = my_str.split("");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
// output:
J
o
e
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[join \(Array.join メソッド\)](#)

String コンストラクタ

```
public String(value:String)
```

新しい String オブジェクトを作成します。

メモ : String オブジェクトよりもストリングリテラルを使用する方が負荷が少なく、一般的により簡単に使用できます。String オブジェクトを使用することに特別な理由がない場合以外は、String クラスのコンストラクタよりもストリングリテラルを使用することをお勧めします。

パラメータ

value: String - 新しい String オブジェクトの初期値。

substr (String.substr メソッド)

```
public substr(start:Number, length:Number) : String
```

ストリング内で start パラメータで指定されたインデックスから length パラメータで指定された文字数までの文字を返します。substr メソッドは、my_str に指定されたストリングを変更せずに、新しいストリングを返します。

パラメータ

start: Number - サブストリングを作成するために使用する my_str 内の開始位置を示す整数。start が負の数値の場合、開始位置はストリングの終わりから決定されます。このとき、-1 が最後の文字です。

length: Number - 作成するサブストリングの文字数。length を指定しないと、サブストリングにはストリングの始めから終わりまでのすべての文字が含まれます。

戻り値

String - 指定されたストリングのサブストリング。

例

次の例では、新しいストリング my_str を作成し、substr() を使用してストリングの 2 つ目の文字を返します。まず正の start パラメータを使用し、次に負の start パラメータを使用します。

```
var my_str:String = new String("Hello world");
var mySubstring:String = new String();
mySubstring = my_str.substr(6,5);
trace(mySubstring); // output: world

mySubstring = my_str.substr(-5,5);
trace(mySubstring); // output: world
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

substring (String.substring メソッド)

public substring(start:Number, end:Number) : String

start パラメータと end パラメータで指定された点の間の文字をストリングとして返します。end パラメータを指定しないと、サブストリングの終わりはストリングの終わりです。start が end と等しい場合、空白のストリングが返されます。start の値が end の値を超えると、2つのパラメータは入れ替えられて関数が実行されます。元の値は変更されません。

パラメータ

start:Number - サブストリングを作成するために使用する my_str の開始位置を示す整数。start パラメータに指定できる値は、0 ~ String.length - 1 です。start パラメータに負数を指定した場合は、0 が使われます。

end:Number - my_str から取り出す最後の文字のインデックスに 1 を加えた整数。end に指定できる値は、1 ~ String.length です。end パラメータで指定されたインデックス位置の文字は、取り出されるストリングには含まれません。このパラメータを省略すると、String.length が使用されます。このパラメータが負の値である場合は、0 が使用されます。

戻り値

String - 指定されたストリングのサブストリング。

例

次の例では、substring() の使用方法を示します。

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(6,11);
trace(mySubstring); // output: world
```

次に、負の start パラメータを使用した場合の例を示します。

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(-5,5);
trace(mySubstring); // output: Hello
```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

toLowerCase (String.toLowerCase メソッド)

public `toLowerCase()` : String

`String` オブジェクトのコピーを返します。すべての大文字が小文字に変換されます。元の値は変更されません。

戻り値

`String` - ストリング。

例

次の例では、すべての文字が大文字になっているストリングを作成した後、`toLowerCase()` を使用して、そのストリングの大文字をすべて小文字に変換したコピーを作成します。

```
var upperCase:String = "LOREM IPSUM DOLOR";
var lowerCase:String = upperCase.toLowerCase();
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "Strings.fla"
ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンの
フォルダに移動してサンプルにアクセスします。
```

関連項目

[toUpperCase \(String.toUpperCase メソッド \)](#)

toString (String.toString メソッド)

public `toString()` : String

プロパティがストリングかどうかに関係なく、オブジェクトのプロパティをストリングとして返します。

戻り値

`String` - ストリング。

例

次の例では、プロパティがストリングであるかどうかに関係なく、オブジェクトのプロパティをすべてリストする大文字のストリングを出力します。

```
var employee:Object = new Object();
employee.name = "bob";
employee.salary = 60000;
employee.id = 284759021;
```

```
var employeeData:String = new String();
for (prop in employee)
{
    employeeData += employee[prop].toString().toUpperCase() + " ";
}
trace(employeeData);

toString() メソッドがこのコードに含まれず、for ループ内の行で
employee[prop].toUpperCase() を使用した場合、出力は "undefined undefined BOB" になります。toString() メソッドを指定すると、目的の出力 "284759021 60000 BOB" が生成されます。
```

toUpperCase (String.toUpperCase メソッド)

public `toUpperCase()` : String
String オブジェクトのコピーを返します。すべての小文字が大文字に変換されます。元の値は変更されません。

戻り値

[String](#) - ストリング。

例

次の例では、すべての文字が小文字になっているストリングを作成した後、`toUpperCase()` を使用して、そのストリングのコピーを作成します。

```
var lowerCase:String = "lorem ipsum dolor";
var upperCase:String = lowerCase.toUpperCase();
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
www.adobe.com/go/learn\_fl\_samples\_jp の "ActionScript" サンプルフォルダにある "Strings.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。
```

関連項目

[toLowerCase \(String.toLowerCase メソッド \)](#)

valueOf (String.valueOf メソッド)

public valueOf() : String

ストリングを返します。

戻り値

[String](#) - ストリングの値。

例

次の例では、String オブジェクトの新しいインスタンスを作成し、valueOf メソッドが String オブジェクトのインスタンスではなく primitive 値への参照を返すことを示します。

```
var str:String = new String("Hello World");
var value:String = str.valueOf();
trace(str instanceof String); // true
trace(value instanceof String); // false
trace(str === value); // false
```

System

[Object](#)

```
|  
+-System
```

```
public class System  
extends Object
```

System クラスには、共有オブジェクトやクリップボードの使用など、ユーザーのコンピュータ上で行われる特定の操作に関連するプロパティが含まれています。System パッケージ内の次に示す特定のクラスには、追加のプロパティとメソッドがあります。capabilities クラス (System.capabilities を参照) および security クラス (System.security を参照)。

関連項目

[capabilities \(System.capabilities\)](#), [security \(System.security\)](#)

プロパティ一覧

オプション	プロパティ	説明
static	useCodepage:Boolean	外部テキストファイルを解析するときに、Unicode を使用するか、または Flash Player を実行するオペレーティングシステムの従来のコードページを使用するかを示すブール値です。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

イベント一覧

イベント	説明
<code>onStatus = function(infoObject: Object) {}</code>	イベントハンドラ。特定のオブジェクトのスーパーイベントハンドラを提 供します。

メソッド一覧

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

onStatus (System.onStatus ハンドラ)

```
onStatus = function(infoObject:Object) {}
```

イベントハンドラ。特定のオブジェクトのスーパーイベントハンドラを提供します。

SharedObject クラスには、情報、ステータス、またはエラーメッセージを提供するために情報オブジェクトを使用する onStatus() イベントハンドラが用意されています。このイベントハンドラに応答するには、情報オブジェクトを処理する関数を作成する必要があります。また、返される情報オブジェクトの形式と内容を知っておく必要があります。

SharedObject.onStatus() メソッドだけでなく、System.onStatus() と呼ばれるスーパー関数も用意されています。この関数は第 2 のエラーメッセージハンドラとして機能します。SharedObject クラスのインスタンスによって情報オブジェクトに "error" の level プロパティが渡された場合、そのインスタンスに onStatus() 関数が定義されていなければ、代わりに System.onStatus() に定義されている関数が使用されます。

パラメータ

`infoObject:Object` - ステータスマッセージに従って定義されるパラメータ。

例

次の例では、特定のクラスに対応する `onStatus()` 関数が存在しない場合に情報オブジェクトを処理する `System.onStatus()` 関数を作成する方法を示します。

```
// Create generic function
System.onStatus = function(genericError:Object){
    // Your script would do something more meaningful here
    trace("An error has occurred. Please try again.");
}
```

関連項目

[onStatus \(SharedObject.onStatus ハンドラ\)](#)

useCodepage (System.useCodepage プロパティ)

public static useCodepage : Boolean

外部テキストファイルを解析するときに、Unicode を使用するか、または Flash Player を実行するオペレーティングシステムの従来のコードページを使用するかを示す布尔値です。

`System.useCodepage` のデフォルト値は `false` です。

- このプロパティを `false` に設定すると、外部テキストファイルは Unicode と解釈されます。これらのファイルは、保存する際に Unicode でエンコードする必要があります。
- このプロパティを `true` に設定すると、Flash Player を実行するオペレーティングシステムの通常のコードページを使用して外部テキストファイルが解釈されます。

外部ファイルとしてロードしたテキスト (`loadVariables()` または `getURL()` ステートメント、`LoadVars` クラスまたは XML クラスを使用) を保存する際には、このテキストファイルを Flash Player で Unicode と認識できるように、Unicode でエンコードする必要があります。外部ファイルを Unicode でエンコードするには、Windows 2000 のメモ帳など、Unicode をサポートするアプリケーションでファイルを保存する必要があります。

Unicode でエンコードされていない外部テキストファイルをインクルードまたはロードする際には、`System.useCodepage` を `true` に設定してください。データをロードする SWF ファイルの最初のフレームの先頭行として、次のコードを追加します。

```
System.useCodepage = true;
```

このコードがあると、外部テキストは、Flash Player を実行するオペレーティングシステムの通常のコードページを使用して解釈されます。一般に、英語の Windows オペレーティングシステムの場合は CP1252、日本語のオペレーティングシステムの場合は Shift-JIS が使用されます。Flash Player 6 以降で `System.useCodepage` を `true` に設定すると、テキストは Flash Player 5 の場合と同様に扱われます。Flash Player 5 では、すべてのテキストは、Flash Player を実行するオペレーティングシステムの通常のコードページを使用して解釈されていました。

`System.useCodepage` を `true` に設定した場合、外部テキストファイル内で使用されている文字が Flash Player を実行するオペレーティングシステムの通常のコードページに含まれていないと、そのテキストは表示されないことに注意してください。たとえば、中国語を含む外部テキストファイルをロードする場合、CP1252 コードページを使用するシステムではこれらの文字を表示できません。CP1252 コードページには中国語が含まれていないためです。

SWF ファイルで使用する外部テキストファイルをすべてのプラットフォームのユーザーが表示できるようにするには、すべての外部テキストファイルを Unicode で保存し、`System.useCodepage` にはデフォルト値の `false` を使用します。これにより、Flash Player 6 以降ではテキストが Unicode として解釈されます。

TextField

```
Object
 |
 +-TextField
```

```
public dynamic class TextField
extends Object
```

`TextField` クラスは、テキストの表示と入力用の領域を作成するために使用されます。SWF ファイルのすべてのダイナミックテキストフィールドおよびテキスト入力フィールドは、`TextField` クラスのインスタンスです。プロパティインスペクタを使用して、テキストフィールドにインスタンス名付けることができます。また、`TextField` クラスのメソッドとプロパティを使用して、ActionScript でテキストフィールドを操作できます。`TextField` インスタンスの名前は、ムービーエクスプローラに表示されます。また、[アクション] パネルの [ターゲットパスの挿入] ダイアログボックスにも表示されます。

テキストフィールドを動的に作成する場合は、`new` 演算子を使用しません。

`MovieClip.createTextField()` を使用します。

`TextField` クラスのメソッドを使用すると、オーサリング時または実行時に作成したダイナミックテキストフィールドやテキスト入力フィールドにテキストを設定、選択、および操作できます。

関連項目

[Object.createTextField \(MovieClip.createTextField メソッド\)](#)

プロパティ一覧

オプション	プロパティ	説明
	<code>_alpha:Number</code>	テキストフィールドのアルファ透明度値を設定または取得します。
	<code>autoSize:Object</code>	テキストフィールドの自動的な拡大 / 縮小および整列を制御します。
	<code>background:Boolean</code>	テキストフィールドに背景の塗りがあるかどうかを指定します。
	<code>backgroundColor:Number</code>	テキストフィールドの背景の色。
	<code>border:Boolean</code>	テキストフィールドに境界線があるかどうかを指定します。
	<code>borderColor:Number</code>	テキストフィールドの境界線の色。
	<code>bottomScroll:Number (読み取り専用)</code>	テキストフィールドの現在の表示範囲で最終行を示す整数(1から始まるインデックス)。
	<code>condenseWhite:Boolean</code>	フィールドをブラウザで表示するときに、HTML テキストフィールド内の余分な空白(スペース、改行など)を削除するかどうかを指定する布尔値。
	<code>embedFonts:Boolean</code>	埋め込みフォントアウトラインを使用してテキストをレンダリングするかどうかを指定する布尔値。
	<code>_height:Number</code>	ピクセル単位で示したテキストフィールドの高さ。
	<code>_highquality:Number</code>	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに <code>TextField._quality</code> を使用します。 現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。
	<code>hscroll:Number</code>	現在の水平スクロール位置を示します。
	<code>html:Boolean</code>	テキストフィールドに HTML 表現があるかどうかを示すフラグ。
	<code>htmlText:String</code>	テキストフィールドが HTML テキストフィールドである場合、このプロパティにはテキストフィールドの内容の HTML 表現が入ります。
	<code>length:Number (読み取り専用)</code>	テキストフィールド内の文字数を示します。
	<code>maxChars:Number</code>	テキストフィールドに入る最大の文字数を示します。
	<code>maxhscroll:Number (読み取り専用)</code>	<code>TextField.hscroll</code> の最大値を示します。

オプション	プロパティ	説明
	<code>maxscroll:Number</code> (読み取り専用)	<code>TextField.scroll</code> の最大値を示します。
	<code>multiline:Boolean</code>	テキストフィールドが複数行テキストフィールドであるかどうかを示します。
	<code>_name:String</code>	テキストフィールドのインスタンス名。
	<code>_parent:MovieClip</code>	現在のテキストフィールドまたはオブジェクトを含むムービークリップまたはオブジェクトへの参照。
	<code>password:Boolean</code>	テキストフィールドがパスワードテキストフィールドであるかどうかを指定します。
	<code>_quality:String</code>	プロパティ (グローバル)。 SWF ファイルに使用するレンダリング品質を設定または取得します。
	<code>_rotation:Number</code>	テキストフィールドの元の位置からの回転角を度単位で指定します。
	<code>scroll:Number</code>	テキストフィールドのテキストの垂直座標を定義します。
	<code>selectable:Boolean</code>	テキストフィールドが選択可能であるかどうかを示す布尔値です。
	<code>_soundbuftime:Number</code>	サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。
	<code>tabEnabled:Boolean</code>	テキストフィールドが自動タブ順に含まれるかどうかを指定します。
	<code>tabIndex:Number</code>	SWF ファイル内のオブジェクトのタブ順をカスタマイズできます。
	<code>_target:String</code> (読み取り専用)	テキストフィールドインスタンスのターゲットパス。
	<code>text:String</code>	テキストフィールドの現在のテキストを示します。
	<code>textColor:Number</code>	テキストフィールドのテキストの色を示します。
	<code>textHeight:Number</code>	テキストの高さを示します。
	<code>textWidth:Number</code>	テキストの幅を示します。
	<code>type:String</code>	テキストフィールドのタイプを指定します。
	<code>_url:String</code> (読み取り専用)	テキストフィールドを作成した SWF ファイルの URL を取得します。
	<code>variable:String</code>	テキストフィールドに関連付けられた変数の名前です。

オプション	プロパティ	説明
	<code>_visible:Boolean</code>	テキストフィールド <code>my_txt</code> が表示されるかどうかを示すブール値です。
	<code>_width:Number</code>	ピクセル単位で示したテキストフィールドの幅。
	<code>wordWrap:Boolean</code>	テキストフィールドのテキストを折り返すかどうかを示すブール値。
	<code>_x:Number</code>	親ムービークリップのローカル座標を基準にしてテキストフィールドの x 座標を設定する整数。
	<code>_xmouse:Number (読み取り専用)</code>	テキストフィールドを基準にしたポインタの x 座標を返します。
	<code>_xscale:Number</code>	テキストフィールドの基準点から適用する、テキストフィールドの水平方向の拡大 / 縮小率(パーセンテージ)を決定します。
	<code>_y:Number</code>	親ムービークリップのローカル座標を基準にしたテキストフィールドの y 座標。
	<code>_ymouse:Number (読み取り専用)</code>	テキストフィールドを基準にしたポインタの y 座標を示します。
	<code>_yscale:Number</code>	テキストフィールドの基準点から適用する、テキストフィールドの垂直方向の拡大 / 縮小率(パーセンテージ)。

Object クラスから継承されるプロパティ

<code>constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)</code>

イベント一覧

イベント	説明
<code>onChanged = function (changedField: TextField) {}</code>	イベントハンドラ / リスナー。テキストフィールドの内容が変更されると、呼び出されます。
<code>onKillFocus = function(newFocus: Object) {}</code>	テキストフィールドがキーボードフォーカスを失うと、呼び出されます。
<code>onScroller = function (scrolledField: TextField) {}</code>	イベントハンドラ / リスナー。テキストフィールドのスクロールプロパティが変わると呼び出されます。
<code>onSetFocus = function(oldFocus: Object) {}</code>	テキストフィールドがキーボードフォーカスを受け取ると、呼び出されます。

メソッド一覧

オプション	シグネチャ	説明
	<code>addListener (listener:Object) : Boolean</code>	TextField イベント通知を受け取るオブジェクトを登録します。
	<code>getDepth() : Number</code>	テキストフィールドの深度を返します。
	<code>getNewTextFormat() : TextFormat</code>	テキストフィールドのテキストフォーマットオブジェクトのコピーを含む TextFormat オブジェクトを返します。
	<code>getTextFormat ([beginIndex:Number], [endIndex:Number]) : TextFormat</code>	文字、文字の範囲または TextField オブジェクト全体に対して TextFormat オブジェクトを返します。
	<code>removeListener (listener:Object) : Boolean</code>	TextField.addListener() でテキストフィールドインスタンスに以前に登録したリスナーオブジェクトを削除します。
	<code>removeTextField() : Void</code>	テキストフィールドを削除します。
	<code>replaceSel(newText: String) : Void</code>	現在の選択内容を newText パラメータの内容に置き換えます。

オプション	シグネチャ	説明
	<code>replaceText (beginIndex:Number, endIndex:Number, newText:String) : Void</code>	指定されたテキストフィールドの beginIndex パラメータと endIndex パラメータで指定した文字範囲を、newText パラメータの内容に置き換えます。
	<code>setNewTextFormat(tf: TextFormat) : Void</code>	テキストフィールドに新しいデフォルトのテキストフォーマットを設定します。
	<code>setTextFormat([begin Index:Number], [endIndex:Number], textFormat: TextFormat) : Void</code>	textFormat パラメータで指定したテキストフォーマットを、テキストフィールド内のテキストの一部またはすべてに適用します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

addListener (TextField.addListener メソッド)

public addListener(listener:Object) : Boolean

TextField イベント通知を受け取るオブジェクトを登録します。このオブジェクトは、onChanged イベントハンドラおよび onScroller イベントハンドラが呼び出されたときにイベント通知を受け取ります。テキストフィールドが変更またはスクロールされると、TextField.onChanged イベントハンドラと TextField.onScroller イベントハンドラが呼び出され、その後リスナーとして登録されているオブジェクトの onChanged イベントハンドラと onScroller イベントハンドラが呼び出されます。複数のオブジェクトをリスナーとして登録することができます。

テキストフィールドからリスナーオブジェクトを削除するには、TextField.removeListener() を呼び出します。

onScroller イベントハンドラと onChanged イベントハンドラには、テキストフィールドインスタンスへの参照がパラメータとして渡されます。イベントハンドラメソッドにパラメータを含めることによって、このデータを受け取ることができます。たとえば、次の例では、onScroller イベントハンドラで txt パラメータを受け取ります。その後、trace ステートメントでこのパラメータを使用して、テキストフィールドのインスタンス名を [出力] パネルに表示します。その後、trace() メソッドでこのパラメータを使用して、テキストフィールドのインスタンス名をログファイルに書き込みます。

```
my_txt.onScroller = function(textfield_txt:TextField) {
    trace(textfield_txt._name+" scrolled");
};
```

パラメータ

listener:Object - onChanged イベントハンドラまたは onScroller イベントハンドラを持つオブジェクト。

戻り値

Boolean -

例

次の例では、テキスト入力フィールド my_txt に対して onChanged ハンドラを定義します。その後、新しいリスナーオブジェクト txtListener を定義し、そのオブジェクト用に onChanged ハンドラを定義します。このハンドラは、テキストフィールド my_txt が変更されたと呼び出されます。このコードの最終行では、TextField.addListener を呼び出し、リスナーオブジェクト txtListener をテキストフィールド my_txt に登録します。これにより、このリスナーは my_txt が変更されたと通知されるようになります。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.type = "input";

my_txt.onChanged = function(textfield_xt:TextField) {
    trace(textfield_txt._name+" changed");
};

var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
    trace(textfield_txt._name+" changed and notified myListener");
};

my_txt.addListener(txtListener);
```

関連項目

[onChanged \(TextField.onChanged ハンドラ\)](#), [onScroller \(TextField.onScroller ハンドラ\)](#), [removeListener \(TextField.removeListener メソッド\)](#)

_alpha (TextField._alpha プロパティ)

public _alpha : Number

テキストフィールドのアルファ透明度値を設定または取得します。有効な値は 0(完全な透明)～100(完全な不透明)です。デフォルト値は 100 です。透明度値は、デバイスフォントを使用するテキストフィールドではサポートされません。テキストフィールドで _alpha 透明度プロパティを使用するには、埋め込みフォントを使用する必要があります。

メモ: このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次のコードは、テキストフィールド my_txt の _alpha プロパティを 20% に設定します。[ライブ ライブ] オプションメニューの [新しいフォント] を選択して、ライブラリに新しいフォントシンボルを作成します。次に、新しいフォントのリンクージを my_font に設定します。フォントシンボルのリンクージを my_font に設定します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";
// where 'my font' is the linkage name of a font in the Library
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.embedFonts = true;
my_txt.text = "Hello World";
my_txt.setTextFormat(my_fmt);
my_txt._alpha = 20;
```

関連項目

[_alpha \(Button._alpha プロパティ\)](#), [_alpha \(MovieClip._alpha プロパティ\)](#)

autoSize (TextField.autoSize プロパティ)

public autoSize : Object

テキストフィールドの自動的な拡大 / 縮小および整列を制御します。autoSize の有効な値は、"none" (デフォルト)、"left"、"right"、"center" の 4 つです。autoSize プロパティを設定する場合、true は "left" を指定するのと同じであり、false は "none" を指定するのと同じです。

autoSize と TextField.wordWrap に指定した値によって、テキストフィールドが左、右、または下に伸縮します。これらのプロパティのデフォルト値は false です。

autoSize が "none" (デフォルト値) または false に設定されていると、サイズ変更は行われません。

`autoSize` を "left" または `true` に設定すると、テキストは左揃えテキストとして扱われます。つまり、テキストフィールドの左側が固定され、單一行テキストフィールドの右側のみが伸縮します。テキストに改行 ("\\n" または "\\r" など) が含まれる場合、テキストの次の行が収まるようにフィールドの下側が拡張されます。`wordWrap` も `true` に設定した場合、テキストフィールドの下側だけが伸縮し、右側は固定されたままになります。

`autoSize` を "right" に設定すると、テキストは右揃えテキストとして扱われます。つまり、テキストフィールドの右側が固定され、單一行テキストフィールドの左側のみが伸縮します。テキストに改行 ("\\n" または "\\r" など) が含まれる場合、テキストの次の行が収まるようにフィールドの下側が拡張されます。`wordWrap` も `true` に設定した場合、テキストフィールドの下側だけが伸縮し、左側は固定されたままになります。

`autoSize` を "center" に設定すると、テキストは中央揃えテキストとして扱われます。つまり、單一行テキストフィールドのサイズ変更を行うと、左右両側が均等に伸縮されます。テキストに改行 ("\\n" または "\\r" など) が含まれる場合、テキストの次の行が収まるようにフィールドの下側が拡張されます。`wordWrap` も `true` に設定した場合、テキストフィールドの下側だけが伸縮し、左右両側は固定されたままになります。

例

次のコードで `autoSize` の値を変えてみてください。指定された値に応じてフィールドのサイズが変化するのがわかります。SWF ファイルの再生中にマウスでクリックすると、各テキストフィールドの "short text" ストリングが、`autoSize` にいくつかの異なる設定が適用された、より長いテキストに置き換わります。

```
this.createTextField("left_txt", 997, 10, 10, 70, 30);
this.createTextField("center_txt", 998, 10, 50, 70, 30);
this.createTextField("right_txt", 999, 10, 100, 70, 30);
this.createTextField("true_txt", 1000, 10, 150, 70, 30);
this.createTextField("false_txt", 1001, 10, 200, 70, 30);

left_txt.text = "short text";
left_txt.border = true;

center_txt.text = "short text";
center_txt.border = true;

right_txt.text = "short text";
right_txt.border = true;

true_txt.text = "short text";
true_txt.border = true;

false_txt.text = "short text";
false_txt.border = true;

// create a mouse listener object to detect mouse clicks
```

```
var myMouseListener:Object = new Object();
// define a function that executes when a user clicks the mouse
myMouseListener.onMouseDown = function() {
    left_txt.autoSize = "left";
    left_txt.text = "This is much longer text";
    center_txt.autoSize = "center";
    center_txt.text = "This is much longer text";
    right_txt.autoSize = "right";
    right_txt.text = "This is much longer text";
    true_txt.autoSize = true;
    true_txt.text = "This is much longer text";
    false_txt.autoSize = false;
    false_txt.text = "This is much longer text";
};
// register the listener object with the Mouse object
Mouse.addListener(myMouseListener);
```

background (TextField.background プロパティ)

public background : Boolean

テキストフィールドに背景の塗りがあるかどうかを指定します。true である場合、テキストフィールドに背景の塗りがあります。false である場合、テキストフィールドには背景の塗りがありません。

例

次の例では、キーボード上の任意のキーを押したときに、オン、オフが切り替わる背景色を持つテキストフィールドを作成します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
my_txt.backgroundColor = 0xFF0000;

var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    my_txt.background = !my_txt.background;
};
Key.addListener(keyListener);
```

backgroundColor (TextField.backgroundColor プロパティ)

public backgroundColor : Number

テキストフィールドの背景の色。デフォルト値は 0xFFFFFFFF (白) です。このプロパティは、現在背景がない場合でも取得または設定できます。ただし、背景の色が表示されるのはテキストフィールドに境界線がある場合のみです。

例

TextField.background の例を参照してください。

関連項目

[background \(TextField.background プロパティ \)](#)

border (TextField.border プロパティ)

public border : Boolean

テキストフィールドに境界線があるかどうかを指定します。true である場合、テキストフィールドに境界線があります。false である場合、テキストフィールドに境界線がありません。

例

次の例では、テキストフィールド my_txt を作成し、境界線のプロパティを true に設定し、フィールド内にテキストを表示します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
```

borderColor (TextField.borderColor プロパティ)

public borderColor : Number

テキストフィールドの境界線の色。デフォルト値は 0x000000 (黒) です。このプロパティは、現在境界線がない場合でも取得または設定できます。

例

次の例では、テキストフィールド my_txt を作成し、境界線のプロパティを true に設定し、フィールド内にテキストを表示します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.borderColor = 0x00FF00;
my_txt.text = "Lorum ipsum";
```

関連項目

[border \(TextField.border プロパティ\)](#)

bottomScroll (TextField.bottomScroll プロパティ)

public bottomScroll : Number (読み取り専用)

テキストフィールドの現在の表示範囲で最終行を示す整数(1から始まるインデックス)。テキストフィールドは、テキストのブロックにかぶせたウィンドウのようなものです。TextField.scroll プロパティは、そのウィンドウに表示されている先頭行を示すインデックス(1から始まるインデックス)です。

TextField.scroll から TextField.bottomScroll までのすべての行が、テキストフィールドに現在表示されています。

例

次の例では、テキストフィールドを作成し、そこにテキストを流し込みます。“my_btn”というインスタンス名のボタンを挿入する必要があります。ボタンをクリックすると、テキストフィールドのcomment_txt フィールドで scroll プロパティと bottomScroll プロパティがトレースされます。

```
this.createTextField("comment_txt", this.getNextHighestDepth(), 0, 0, 160, 120);
comment_txt.html = true;
comment_txt.selectable = true;
comment_txt.multiline = true;
comment_txt.wordWrap = true;
comment_txt.htmlText = "<b>What is hexadecimal?</b><br>" +
    + "The hexadecimal color system uses six digits to represent color values. " +
    + "Each digit has sixteen possible values or characters. The characters range" +
    + " from 0 to 9 and then A to F. Black is represented by (#000000) and white, " +
    + "at the opposite end of the color system, is (#FFFFFF).";
my_btn.onRelease = function() {
    trace("scroll: "+comment_txt.scroll);
    trace("bottomScroll: "+comment_txt.bottomScroll);
};
```

condenseWhite (TextField.condenseWhite プロパティ)

public condenseWhite : Boolean

フィールドをブラウザで表示するときに、HTML テキストフィールド内の余分な空白(スペース、改行など)を削除するかどうかを指定する布尔値。デフォルト値は false です。

この値を true に設定した場合は、テキストフィールド内で改行を指定するときに
 や<P>などの標準の HTML コマンドを使用する必要があります。

テキストフィールドの .html が false である場合、このプロパティは無視されます。

例

次の例では、2つのテキストフィールド first_txt および second_txt を作成します。2番目のテキストフィールドから空白が削除されます。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
var my_str:String = "Hello\tWorld\nHow are you?\t\t\tEnd";  
  
this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 120);  
first_txt.html = true;  
first_txt.multiline = true;  
first_txt.wordWrap = true;  
first_txt.condenseWhite = false;  
first_txt.border = true;  
first_txt.htmlText = my_str;  
  
this.createTextField("second_txt", this.getNextHighestDepth(), 180, 10, 160, 120);  
second_txt.html = true;  
second_txt.multiline = true;  
second_txt.wordWrap = true;  
second_txt.condenseWhite = true;  
second_txt.border = true;  
second_txt.htmlText = my_str;
```

関連項目

[html \(TextField.html プロパティ\)](#)

embedFonts (TextField.embedFonts プロパティ)

public embedFonts : Boolean

埋め込みフォントアウトラインを使用してテキストをレンダリングするかどうかを指定する布尔値。値が true の場合、Flash Lite は埋め込みフォントアウトラインを使用してテキストフィールドをレンダリングします。値が false の場合、Flash Lite はデバイスフォントを使用してテキストフィールドをレンダリングします。

テキストフィールドの embedFonts を true に設定する場合、テキストフィールドに適用される TextFormat オブジェクトの font プロパティ経由で、そのテキストのフォントを指定する必要があります。指定されたフォントが対応するリンクエージェンシーを持たないライブラリに存在しない場合、テキストは表示されません。

メモ: このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

この例では、ダイナミックテキストフィールド my_txt を作成してから、次の ActionScript を使用してフォントの埋め込みとテキストフィールドの回転を行う必要があります。ストリング my_font は、リンクエージ識別子名 my_font を使用してライブラリ内のフォントシンボルを参照します。この例では、my_font というライブラリ内に、リンクエージプロパティで、[識別子] が my_font に設定され、[ActionScript に書き出し] と [最初のフレームに書き出し] が選択されているフォントシンボルがあるものとします。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

getDepth (TextField.getDepth メソッド)

public getDepth() : Number

テキストフィールドの深度を返します。

戻り値

Number - 整数。

例

次の例は、それぞれ異なる深度にあるテキストフィールドを示しています。ステージ上にダイナミックテキストフィールドを作成します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加すると、実行時に 2 つのテキストフィールドが動的に作成され、それぞれの深度が表示されます。

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
        var this_txt:TextField = this[prop];
        trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

getNewTextFormat (TextField.getNewTextFormat メソッド)

```
public getNewTextFormat() : TextFormat
```

テキストフィールドのテキストフォーマットオブジェクトのコピーを含む TextFormat オブジェクトを返します。テキストフォーマットオブジェクトは、新しく挿入するテキスト（ユーザーが入力したテキストなど）に適用されるフォーマットです。getNewTextFormat() を呼び出すと、すべてのプロパティが定義された TextFormat オブジェクトが返されます。null のプロパティはありません。

戻り値

[TextFormat](#) - TextFormat オブジェクト。

例

次の例では、指定されたテキストフィールドの (my_txt) テキストフォーマットオブジェクトを表示します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
var my_fmt:TextFormat = my_txt.getNewTextFormat();
trace("TextFormat has the following properties:");
for (var prop in my_fmt) {
    trace(prop+": "+my_fmt[prop]);
}
```

getTextFormat (TextField.getTextFormat メソッド)

```
public getTextFormat([beginIndex:Number], [endIndex:Number]) : TextFormat
```

文字、文字の範囲または TextField オブジェクト全体に対して TextFormat オブジェクトを返します。

■ シンタックス 1:

```
my_textField.getTextFormat()
```

テキストフィールド内すべてのテキストに関するフォーマット情報を含む TextFormat オブジェクトを返します。テキストフィールド内のすべてのテキストに共通するプロパティのみが結果の TextFormat オブジェクトに設定されます。混在型のプロパティ（テキストに複数の異なる値が混在する場合）は、その値が null に設定されます。

■ シンタックス 2:

```
my_textField.getTextFormat(beginIndex:Number)
```

テキストフィールドで beginIndex の位置でのテキストフォーマットのコピーを含む TextFormat オブジェクトを返します。

■ シンタックス 3:

```
my_(extField.getTextFormat(beginIndex:Number,endIndex:Number)  
beginIndex から endIndex までの範囲のテキストに関するフォーマット情報を含む TextFormat  
オブジェクトを返します。指定された範囲内のすべてのテキストに共通するプロパティのみが結  
果の TextFormat オブジェクトに設定されます。混在型のプロパティ（範囲内に複数の異なる値  
が混在する場合）は、その値が null に設定されます。
```

パラメータ

beginIndex:Number (オプション) - ストリング内の文字を指定する整数。beginIndex および
endIndex を指定しない場合、返される TextFormat オブジェクトの対象は TextField 全体になります。
endIndex:Number (オプション) - テキスト範囲の終了位置を指定する整数。beginIndex を指定
して endIndex を指定しない場合、返される TextFormat の対象は beginIndex で指定された 1つ
の文字になります。

戻り値

[TextFormat](#) - オブジェクト。

例

次の ActionScript は、実行時に作成されたテキストフィールドの全フォーマット情報を表示します。

```
this.createTextField("dyn_txt", this.getNextHighestDepth(), 0, 0, 100, 200);  
dyn_txt.text = "Frank";  
dyn_txt.setTextFormat(new TextFormat());  
var my_fmt:TextFormat = dyn_txt.getTextFormat();  
for (var prop in my_fmt) {  
    trace(prop+": "+my_fmt[prop]);  
}
```

関連項目

[getNewTextFormat](#) ([TextField.getNewTextFormat メソッド](#)), [setNewTextFormat](#)
([TextField.setNewTextFormat メソッド](#)), [setTextFormat](#) ([TextField.setTextFormat
メソッド](#))

_height (TextField._height プロパティ)

public _height : Number

ピクセル単位で示したテキストフィールドの高さ。

例

次の例では、テキストフィールドの高さと幅のプロパティを設定します。

```
my_txt._width = 200;  
my_txt._height = 200;
```

_highquality (TextField._highquality プロパティ)

public _highquality : Number

非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに TextField._quality を使用します。

現在の SWF ファイルに適用されるアンチエイリアスのレベルを指定します。ビットマップスミージングを常にオンにして最高品質を適用するには、2(最高品質) を指定します。アンチエイリアス処理を適用するには、1(高品質) を指定します。SWF ファイルにアニメーションが含まれない場合は、ビットマップは滑らかになります。これはデフォルト値です。アンチエイリアス処理を避けるには、0(低品質) を指定します。

関連項目

[_quality \(TextField._quality プロパティ \)](#)

hscroll (TextField.hscroll プロパティ)

public hscroll : Number

現在の水平スクロール位置を示します。hscroll プロパティが 0 である場合、テキストは水平にスクロールされません。

垂直スクロールの単位は行数ですが、水平スクロールの単位はピクセル数です。水平スクロールをピクセル単位で指定するのは、一般的に使用されるフォントのほとんどがプロポーショナルフォントであり、文字の幅が一定でないためです。垂直スクロールの場合は、通常、1行のテキストの一部だけ表示されるよりも行全体が表示されることが好まれるため、行単位でスクロールします。1行の中に複数のフォントが存在する場合でも、使用されている最大のフォントに合わせて行の高さが調整されます。

メモ : hscroll プロパティは、垂直スクロールプロパティ TextField.scroll のように 1 から始まるのではなく、0 から始まります。

例

次の例では、2つのボタン scrollLeft_btn および scrollRight_btn を使用して、テキストフィールド my_txt を水平方向にスクロールします。スクロール量は、テキストフィールド scroll_txt に表示されます。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 160, 22);
my_txt.border = true;
my_txt.multiline = false;
my_txt.wordWrap = false;
my_txt.text = "Lorem ipsum dolor sit amet, consectetur adipiscing...";

scrollLeft_btn.onRelease = function() {
    my_txt.hscroll -= 10;
    scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
scrollRight_btn.onRelease = function() {
    my_txt.hscroll += 10;
    scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
```

関連項目

[maxhscroll \(TextField.maxhscroll プロパティ\)](#), [scroll \(TextField.scroll プロパティ\)](#)

html (TextField.html プロパティ)

```
public html : Boolean
```

テキストフィールドに HTML 表現があるかどうかを示すフラグ。html プロパティが true である場合、テキストフィールドは HTML テキストフィールドです。html プロパティが false である場合、テキストフィールドは HTML 以外のテキストフィールドです。

例

次の例では、html プロパティを true に設定するテキストフィールドを作成します。このテキストフィールドには、HTML 形式のテキストが表示されます。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

関連項目

[htmlText \(TextField.htmlText プロパティ\)](#)

htmlText (TextField.htmlText プロパティ)

```
public htmlText : String
```

テキストフィールドが HTML テキストフィールドである場合、このプロパティにはテキストフィールドの内容の HTML 表現が入ります。テキストフィールドが HTML テキストフィールドではない場合、その動作は text プロパティと同じです。テキストフィールドを HTML テキストフィールドとして指定するには、プロパティインスペクタを使用するか、テキストフィールドの html プロパティを true に設定します。

例

次の例では、html プロパティを true に設定するテキストフィールドを作成します。このテキストフィールドには、HTML 形式のテキストが表示されます。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "< this is bold text >";
```

関連項目

[html \(TextField.html プロパティ \)](#)

length (TextField.length プロパティ)

```
public length : Number (読み取り専用)
```

テキストフィールド内の文字数を示します。このプロパティは、text.length 同じ値をより速く返します。タブ(\t)などの文字も1文字としてカウントされます。

例

次の例では、現在の日付を表示するテキストフィールド date_txt 内の文字数を出力します。

```
var today:Date = new Date();
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.text = today.toString();
trace(date_txt.length);
```

maxChars (TextField.maxChars プロパティ)

```
public maxChars : Number
```

テキストフィールドに入る最大の文字数を示します。スクリプトは maxChars の許容数を超えるテキストを挿入できます。maxChars プロパティで指定されるのは、ユーザーが入力できるテキストの量だけです。このプロパティの値が null である場合、ユーザーが入力できるテキストの量には制限がありません。

例

次の例では、フィールド内にユーザーが2文字まで入力できるテキストフィールド age_txt を作成します。

```
this.createTextField("age_txt", this.getNextHighestDepth(), 10, 10, 30, 22);
age_txt.type = "input";
age_txt.border = true;
age_txt.maxChars = 2;
```

maxhscroll (TextField.maxhscroll プロパティ)

public maxhscroll : Number (読み取り専用)

TextField.hscroll の最大値を示します。

例

TextField.hscroll の例を参照してください。

maxscroll (TextField.maxscroll プロパティ)

public maxscroll : Number (読み取り専用)

TextField.scroll の最大値を示します。

例

次の例では、スクロールテキストフィールド my_txt の最大値を設定します。テキストフィールドをスクロールする2つのボタン、scrollUp_btn および scrollDown_btn を作成します。次のActionScript をFLAファイルまたはActionScriptファイルに追加します。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    diam nonummy nibh "
    + "euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

multiline (TextField.multiline プロパティ)

```
public multiline : Boolean
```

テキストフィールドが複数行テキストフィールドであるかどうかを示します。値が true である場合は複数行テキストフィールド、値が false である場合は單一行テキストフィールドです。

例

次の例では、複数行テキストフィールド myText を作成します。

```
this.createTextField("myText", this.getNextHighestDepth(), 10, 30, 110, 100);
myText.text = "Flash is an authoring tool that designers and developers use to
    create presentations,
applications, and other content that enables user interaction.";
myText.border = true;
myText.wordWrap = true;
myText.multiline = true;
```

_name (TextField._name プロパティ)

```
public _name : String
```

テキストフィールドのインスタンス名。

例

次の例は、それぞれ異なる深度にあるテキストフィールドを示しています。ステージ上にダイナミックテキストフィールドを作成します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加すると、実行時に 2 つのテキストフィールドが動的に作成され、それぞれの深度が [出力] パネルに表示されます。

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
        var this_txt:TextField = this[prop];
        trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

ドキュメントをテストするときには、インスタンス名と深度が [出力] パネルに表示されます。ドキュメントをテストするときには、インスタンス名と深度がログファイルに書き込まれます。

onChanged (TextField.onChanged ハンドラ)

```
onChanged = function(changedField:TextField) {}
```

イベントハンドラ / リスナー。テキストフィールドの内容が変更されると、呼び出されます。デフォルトは `undefined` です。スクリプトで定義できます。

`onChanged` ハンドラには、パラメータとしてテキストフィールドインスタンスへの参照が渡されます。イベントハンドラメソッドにパラメータを含めることによって、このデータを受け取ることができます。たとえば、次の例では、`onChanged` イベントハンドラで `textfield_txt` パラメータを受け取ります。その後、`trace()` ステートメントでこのパラメータを使用して、テキストフィールドのインスタンス名を [出力] パネルに表示します。

```
this.createTextField("myInputText_txt", 99, 10, 10, 300, 20);
myInputText_txt.border = true;
myInputText_txt.type = "input";
```

```
myInputText_txt.onChanged = function(textfield_txt:TextField) {
    trace("the value of "+textfield_txt._name+" was changed. New value is:
        "+textfield_txt.text);
};
```

`onChanged` ハンドラは、ユーザーの操作によって変更が生じた場合にのみ呼び出されます。たとえば、ユーザーがキーボードで入力した場合や、マウスを使用してテキストフィールドの内容を変更した場合、メニューアイテムを選択した場合などです。プログラムによってテキストフィールドが変更されても、テキストフィールドに加えられる変更是コードから識別できるため、`onChanged` イベントはトリガされません。

パラメータ

`changedField:TextField` - イベントをトリガするフィールド。

関連項目

[TextFormat, setNewTextFormat \(TextField.setNewTextFormat メソッド\)](#)

onKillFocus (TextField.onKillFocus ハンドラ)

```
onKillFocus = function(newFocus:Object) {}
```

テキストフィールドがキーボードフォーカスを失うと、呼び出されます。`onKillFocus` メソッドは、1つのパラメータ `newFocus` を受け取ります。このパラメータは、フォーカスを受け取る新しいオブジェクトを表すオブジェクトです。フォーカスを受け取るオブジェクトがない場合、`newFocus` の値は `null` です。

パラメータ

`newFocus:Object` - フォーカスを受け取るオブジェクト。

例

次の例では、2つのテキストフィールド first_txt および second_txt を作成します。1つのテキストフィールドにフォーカスを置くと、フォーカスを置かれたテキストフィールドに関する情報と、フォーカスを失ったテキストフィールドに関する情報が、[出力] パネルに表示されます。

```
this.createTextField("first_txt", 1, 10, 10, 300, 20);
first_txt.border = true;
first_txt.type = "input";
this.createTextField("second_txt", 2, 10, 40, 300, 20);
second_txt.border = true;
second_txt.type = "input";
first_txt.onKillFocus = function(newFocus:Object) {
    trace(this._name+" lost focus. New focus changed to: "+newFocus._name);
};
first_txt.onSetFocus = function(oldFocus:Object) {
    trace(this._name+" gained focus. Old focus changed from: "+oldFocus._name);
}
```

関連項目

[onSetFocus \(TextField.onSetFocus ハンドラ\)](#)

onScroller (TextField.onScroller ハンドラ)

```
onScroller = function(scrolledField:TextField) {}
```

イベントハンドラ / リスナー。テキストフィールドのスクロールプロパティが変わると呼び出されます。

onScroller ハンドラには、パラメータとしてテキストフィールドインスタンスへの参照が渡されます。イベントハンドラメソッドにパラメータを含めることによって、このデータを受け取ることができます。たとえば、次の例では、onScroller イベントハンドラで my_txt パラメータを受け取ります。その後、trace() ステートメントでこのパラメータを使用して、テキストフィールドのインスタンス名を [出力] パネルに表示します。

```
myTextField.onScroller = function (my_txt:TextField) {
    trace (my_txt._name + " scrolled");
};
```

TextField.onScroller イベントハンドラは、一般的にはスクロールバーを実装するために使用されます。通常スクロールバーには、テキストフィールドにおける現在の水平スクロール位置または垂直スクロール位置を示すサムなどのインジケータがあります。テキストフィールドはマウスとキーボードを使用して操作し、スクロール位置を変更することができます。このようなユーザー操作によってスクロール位置が変更されたかどうかを、スクロールバーのコードに通知する必要があります。そのため TextField.onScroller を使用します。

`onScroller` は、ユーザーによるテキストフィールドの操作、またはプログラムによって、スクロール位置が変更された場合に呼び出されます。`onChanged` ハンドラは、ユーザーの操作によって変更が生じた場合にのみ発行されます。コードの一部によってスクロール位置が変更されても、スクローラバーのコードに関連付けがない場合、通知がないとスクローラバーのコードではスクロール位置の変更が認識されないため、この 2 つのオプションが必要になります。

パラメータ

`scrolledField:TextField` - スクロール位置が変更された `TextField` オブジェクトへの参照。

例

次の例では、テキストフィールド `my_txt` を作成し、2 つのボタン、`scrollUp_btn` および `scrollDown_btn` を使用して、このテキストフィールドのコンテンツをスクロールします。`onScroller` イベントハンドラが呼び出されると、`trace` ステートメントを使用して [出力] パネルに情報を表示します。インスタンス名 `scrollUp_btn` および `scrollDown_btn` を付けて 2 つのボタンを作成し、次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;

for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
diam "
    + "nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
};
my_txt.onScroller = function() {
    trace("onScroller called");
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

関連項目

[hscroll \(TextField.hscroll プロパティ\)](#), [maxhscroll \(TextField.maxhscroll プロパティ\)](#), [maxscroll \(TextField.maxscroll プロパティ\)](#), [scroll \(TextField.scroll プロパティ\)](#)

onSetFocus (TextField.onSetFocus ハンドラ)

```
onSetFocus = function(oldFocus:Object) {}
```

テキストフィールドがキーボードフォーカスを受け取ると、呼び出されます。oldFocus パラメータは、フォーカスを失うオブジェクトです。たとえば、Tab キーを押して入力フォーカスをボタンからテキストフィールドに移動すると、oldFocus にボタンインスタンスが入ります。前にフォーカスがあったオブジェクトが存在しない場合、oldFocus には null 値が入ります。

パラメータ

oldFocus:Object - フォーカスを失うオブジェクト。

例

TextField.onKillFocus の例を参照してください。

関連項目

[onKillFocus \(TextField.onKillFocus ハンドラ \)](#)

_parent (TextField._parent プロパティ)

```
public _parent : MovieClip
```

現在のテキストフィールドまたはオブジェクトを含むムービークリップまたはオブジェクトへの参照。現在のオブジェクトとは、_parent を参照する ActionScript コードがあるオブジェクトです。

現在のテキストフィールドよりも上位のムービークリップまたはオブジェクトへの相対パスを指定するには、_parent を使用します。_parent を使用して表示リストの複数のレベルを上位に移動するには、次のようにします。

```
_parent._parent._alpha = 20;
```

例

次の ActionScript は、2つのテキストフィールドを作成し、各オブジェクトの _parent に関する情報を出力します。最初のテキストフィールド first_txt は、メインタイムラインに作成されます。

2番目のテキストフィールド second_txt は、ムービークリップ holder_mc 内に作成されます。

```
this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
first_txt.border = true;
trace(first_txt.name+"'s _parent is: "+first_txt._parent);

this.createEmptyMovieClip("holder_mc", this.getNextHighestDepth());
holder_mc.createTextField("second_txt", holder_mc.getNextHighestDepth(), 10, 40,
160, 22);
holder_mc.second_txt.border = true;
```

```
trace(holder_mc.second_txt._name+"'s _parent is:  
"+holder_mc.second_txt._parent);
```

次の情報が[出力]パネルに表示されます。次の情報がログファイルに書き込まれます。

```
first_txt'(_parent is: _level0  
second_txt's _parent is: _level0.holder_mc
```

関連項目

[_parent \(Button._parent プロパティ\)](#), [_parent \(MovieClip._parent プロパティ\)](#),
[_root プロパティ](#)

password (TextField.password プロパティ)

```
public password : Boolean
```

テキストフィールドがパスワードテキストフィールドであるかどうかを指定します。password の値が true である場合、テキストフィールドはパスワードテキストフィールドです。ユーザーがパスワードの入力を終えて [OK] をクリックすると、テキストフィールドに入力された文字はアスタリスクで隠されます。false である場合、テキストフィールドはパスワードテキストフィールドではありません。パスワードモードを有効にすると、[カット] コマンドと [コピー] コマンド、およびそれに対応するキーボードショートカットが機能しなくなります。このセキュリティ機能により、ユーザーの不在時にキーボードショートカットを使用して悪質なユーザーがパスワードが盗むことを防止できます。

例

次の例では、2つのテキストフィールド username_txt および password_txt を作成します。両方のテキストフィールドにテキストが入力されますが、password_txt では password プロパティが true に設定されています。ユーザーがパスワードを入力して [OK] をクリックすると、password_txt フィールドに入力した文字はアスタリスクで表示されます。

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100, 22);  
username_txt.border = true;  
username_txt.type = "input";  
username_txt.maxChars = 16;  
username_txt.text = "hello";  
  
this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100, 22);  
password_txt.border = true;  
password_txt.type = "input";  
password_txt.maxChars = 16;  
password_tx(.password = true;  
password_txt.text = "world";
```

_quality (TextField._quality プロパティ)

```
public _quality : String
```

プロパティ (グローバル) 。SWF ファイルに使用するレンダリング品質を設定または取得します。デバイスフォントは常にエイリアス処理されるため、_quality プロパティには影響されません。

メモ : このプロパティを TextField オブジェクトに対して指定することができますが、実際にはグローバルプロパティであるので、単に _quality という形で値を指定することもできます。詳細については、_quality プロパティを参照してください。

_quality プロパティは、次の値に設定できます。

- "LOW" 低いレンダリング品質。グラフィックスはアンチエイリアス処理されず、ピットマップはスムージングされません。
- "MEDIUM" 普通のレンダリング品質。グラフィックスは 2×2 ピクセルグリッドを使用してアンチエイリアス処理されますが、ピットマップはスムージングされません。この品質は、テキストを含まないムービーに適しています。
- "HIGH" 高いレンダリング品質。グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理されます。ピットマップは、ムービーが静的なものである場合は、スムージングされます。Flash で使用されるデフォルトのレンダリング品質です。
- "BEST" 非常に高いレンダリング品質。グラフィックスは 4×4 ピクセルグリッドを使用してアンチエイリアス処理され、ピットマップは常にスムージングされます。

メモ : このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次の例では、レンダリング品質を LOW に設定します。

```
my_txt._quality = "LOW";
```

関連項目

[_quality プロパティ](#)

removeListener (TextField.removeListener メソッド)

```
public removeListener(listener:Object) : Boolean
```

TextField.addListener() でテキストフィールドインスタンスに以前に登録したリスナーオブジェクトを削除します。

パラメータ

listener: Object - TextField.onChanged または TextField.onScroller から通知を受け取らなくなるオブジェクト。

戻り値

`Boolean` - `listener` が正常に削除された場合は `true` を返します。`listener` が正常に削除されなかった場合 (`listener` が `TextField` オブジェクトのリスナーリストにない場合など) は `false` を返します。

例

次の例では、テキスト入力フィールド `my_txt` を作成します。ユーザーがこのテキストフィールドに入力すると、テキストフィールド内の文字数に関する情報が [出力] パネルに表示されます。ユーザーがこのテキストフィールドに入力すると、テキストフィールド内の文字数に関する情報がログファイルに書き込まれます。ユーザーが `removeListener_btn` インスタンスをクリックした場合は、リスナーが削除され、情報が表示されなくなります。ユーザーが `removeListener_btn` インスタンスをクリックした場合は、リスナーが削除され、情報がログファイルに書き込まれなくなります。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
my_txt.border = true;
my_txt.type = "input";

var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
    trace(textfield_txt+" changed. Current length is: "+textfield_txt.length);
};
my_txt.addListener(txtListener);

removeListener_btn.onRelease = function() {
    trace("Removing listener...");
    if (!my_txt.removeListener(txtListener)) {
        trace("Error! Unable to remove listener");
    }
};
```

removeTextField (TextField.removeTextField メソッド)

```
public removeTextField() : Void
```

テキストフィールドを削除します。この操作を実行できるのは、`MovieClip.createTextField()` で作成したテキストフィールドだけです。このメソッドを呼び出すと、テキストフィールドは削除されます。このメソッドは `MovieClip.removeMovieClip()` と似ています。

例

次の例では、`remove_btn` インスタンスをクリックしてステージから削除できるテキストフィールドを作成します。`remove_btn` という名前を付けてボタンを作成し、次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
my_txt.text = new Date().toString();
my_txt.border = true;

remove_btn.onRelease = function() {
    my_txt.removeTextField();
};
```

replaceSel (TextField.replaceSel メソッド)

public replaceSel(newText:String) : Void

現在の選択内容を newText パラメータの内容に置き換えます。テキストは、現在のデフォルトの文字フォーマットとデフォルトの段落フォーマットを使用して、現在の選択内容の位置に挿入されます。テキストフィールドが HTML テキストフィールドである場合でも、テキストは HTML として扱われません。

replaceSel() メソッドを使用すると、他の部分のテキストの文字フォーマットおよび段落フォーマットを損なわずにテキストを挿入および削除できます。

メモ: replaceSel() メソッドを呼び出す前に、Selection.setFocus() メソッドを使用してフィールドをフォーカスする必要があります。

パラメータ

newText: String - ストリング。

例

次のコード例では、テキストが設定された複数行のテキストフィールドをステージ上に作成します。テキストの一部を選択し、テキストフィールドの上で右クリック (Windows の場合) または Control キーを押しながらクリック (Macintosh の場合) すると、コンテキストメニューから "Enter current date" を選択できます。これにより、選択したテキストを現在の日付に置き換える関数が呼び出されます。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.text = "Select some sample text from the text field and then right-click/
control click "
    + "and select 'Enter current date' from the context menu to replace the "
    + "currently selected text with the current date.";

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new MenuItem("Enter current date", enterDate));
function enterDate(obj:Object, menuItem:MenuItem) {
    var today_str:String = new Date().toString();
```

```
    var date_str:String = today_str.split(" ", 3).join(" ");
    my_txt.replaceSel(date_str);
}
my_txt.menu = my_cm;
```

関連項目

[setFocus \(Selection.setFocus メソッド\)](#)

replaceText (TextField.replaceText メソッド)

public replaceText(beginIndex:Number, endIndex:Number, newText:String) : Void

指定されたテキストフィールドの beginIndex パラメータと endIndex パラメータで指定した文字範囲を、newText パラメータの内容に置き換えます。

パラメータ

beginIndex:Number - 置換範囲の開始インデックスの値。

endIndex:Number - 置換範囲の終了インデックスの値。

newText:String - 指定された文字範囲の置き換えに使用されるテキスト。

例

次に例では、my_txt というテキストフィールドを作成し、テキスト dog@house.net をこのフィールドに割り当てます。indexOf() メソッドを使用して、指定されたシンボル (@) が最初に出現する場所を探します。このシンボルが見つかると、指定されたテキスト（インデックス 0 とシンボルの間）がストリング bird に置き換わります。このシンボルが見つからなかった場合は、[出力] パネルにエラーメッセージが表示されます。このシンボルが見つからなかった場合は、エラーメッセージがログファイルに書き込まれます。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 22);
my_txt.autoSize = true;
my_txt.text = "dog@house.net";

var symbol:String = "@";
var symbolPos:Number = my_txt.text.indexOf(symbol);
if (symbolPos>-1) {
    my_txt.replaceText(0, symbolPos, "bird");
} else {
    trace("symbol '" + symbol + "' not found.");
}
```

_rotation (TextField._rotation プロパティ)

public _rotation : Number

テキストフィールドの元の位置からの回転角を度単位で指定します。時計回りに回転させる場合は0～180の値を指定します。反時計回りに回転させる場合は0～-180の値を指定します。この範囲を超える値は、360の倍数を加算または減算され、範囲内に収まる値になるように調整されます。たとえば、my_txt._rotation = 450というステートメントはmy_txt._rotation = 90と同義です。デバイスフォントを使用しているテキストフィールドに対する回転値の使用はサポートされていません。テキストフィールドで _rotation を使用するには、埋め込みフォントを使用する必要があります。

メモ: このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

この例では、ダイナミックテキストフィールド my_txt を作成してから、次の ActionScript を使用してフォントの埋め込みとテキストフィールドの回転を行う必要があります。ストリング my_font は、リンクエージェント別子 my_font を使用してライブラリ内のフォントシンボルを参照します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

テキストフィールドにその他のフォーマットを適用するには、TextFormat class を使用します。

関連項目

[_rotation \(Button._rotation プロパティ\)](#), [_rotation \(MovieClip._rotation プロパティ\)](#), [TextFormat](#)

scroll (TextField.scroll プロパティ)

public scroll : Number

テキストフィールドのテキストの垂直座標を定義します。scroll プロパティは、長い文節内の特定の段落にユーザーを誘導したり、スクロールテキストフィールドを作成したりする場合に便利です。このプロパティは、取得および変更が可能です。

垂直スクロールの単位は行数ですが、水平スクロールの単位はピクセル数です。水平スクロールをピクセル単位で指定するのは、一般的に使用されるフォントのほとんどがプロポーショナルフォントであり、文字の幅が一定でないためです。垂直スクロールの場合は、通常、1行のテキストの一部だけ表示されるよりも行全体が表示されることが好まれるため、行単位でスクロールします。1行の中に複数のフォントが存在する場合でも、使用されている最大のフォントに合わせて行の高さが調整されます。

例

次の例では、スクロールテキストフィールド my_txt の最大値を設定します。テキストフィールドをスクロールする2つのボタン、scrollUp_btn および scrollDown_btn を作成します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
diam nonummy "
    + "nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

関連項目

[hscroll \(TextField.hscroll プロパティ\)](#), [maxscroll \(TextField.maxscroll プロパティ\)](#)

selectable (TextField.selectable プロパティ)

public selectable : Boolean

テキストフィールドが選択可能であるかどうかを示す布尔値です。値が true の場合、テキストは選択可能です。selectable プロパティを使用することで、テキストフィールドが選択可能かどうかを指定できますが、編集可能かどうかは指定できません。ダイナミックテキストフィールドは、編集可能でない場合でも選択可能にすることができます。ダイナミックテキストフィールドが選択可能でない場合は、そのフィールド内のテキストを選択できません。

`selectable` を `false` に設定すると、テキストフィールド内のテキストはマウスやキーボードからの選択コマンドに応答しなくなり、[コピー] コマンドを使用してテキストをコピーすることができなくなります。`selectable` を `true` に設定すると、テキストフィールド内のテキストはマウスやキーボードを使用して選択できるようになります。テキスト入力フィールドではなくダイナミックテキストフィールドの場合でも、この方法でテキストを選択できます。また [コピー] コマンドを使用してテキストをコピーすることもできます。

メモ : このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次の例では、現在の日付と時刻を反映して常に更新される、選択可能なテキストフィールドを作成します。

```
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.selectable = true;

( var date_interval:Number = setInterval(updateTime, 500, date_txt);
function updateTime(my_txt:TextField) {
    my_txt.text = new Date().toString();
}
```

setNewTextFormat (TextField.setNewTextFormat メソッド)

```
public setNewTextFormat(tf:TextFormat) : Void
```

テキストフィールドに新しいデフォルトのテキストフォーマットを設定します。新しいデフォルトのテキストフォーマットは、新しく挿入するテキスト（ユーザーが入力したテキストなど）に使用される新しいテキストフォーマットです。テキストを挿入すると、新しく挿入されたテキストにその新しいデフォルトのテキストフォーマットが割り当てられます。

新しいデフォルトのテキストフォーマットは、`TextFormat` オブジェクトである `textFormat` を使用して指定します。

パラメータ

`tf:TextFormat` - `TextFormat` オブジェクト。

例

次の例では、実行時に新しいテキストフィールド (`my_txt`) を作成し、いくつかのプロパティを設定します。新しく挿入されたテキストのフォーマットが適用されます。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
```

```
my_fmt.color = 0xFF9900;  
  
this.createTextField("my_txt", 999, 0, 0, 400, 300);  
my_txt.wordWrap = true;  
my_txt.multiline = true;  
my_txt.border = true;  
my_txt.type = "input";  
my_txt.setNewTextFormat(my_fmt);  
my_txt.text = "Oranges are a good source of vitamin C";
```

関連項目

[getNewTextFormat \(TextField.getNewTextFormat メソッド\)](#), [getTextFormat \(TextField.getTextFormat メソッド\)](#), [setTextFormat \(TextField.setTextFormat メソッド\)](#)

setTextFormat (TextField.setTextFormat メソッド)

```
public setTextFormat([beginIndex:Number], [endIndex:Number],  
textFormat:TextFormat) : Void
```

textFormat パラメータにより指定されたテキストフォーマットを、テキストフィールド内のテキストの一部または全体に適用します。textFormat には、目的のテキストフォーマット変更を指定する TextFormat オブジェクトを指定する必要があります。textFormat の null 以外のプロパティのみがテキストフィールドに適用されます。textFormat で null に設定されているプロパティは適用されません。デフォルトで、新しく作成された TextFormat オブジェクトのプロパティはすべて null に設定されます。

TextFormat オブジェクトのフォーマット情報には、文字レベルフォーマットと段落レベルフォーマットの 2 種類があります。テキストフィールド内の各文字にも、フォント名やフォントサイズ、ボーランド、イタリックなどの文字固有のフォーマット設定があります。

段落の場合は、段落の最初の文字を調べて段落全体のフォーマット設定を決定します。段落のフォーマット設定には、左マージン、右マージン、インデントなどがあります。

setTextFormat() メソッドは、テキストフィールドの各文字、文字の範囲、またはテキスト全体に適用するテキストフォーマットを変更します。

■ シンタックス 1:

```
my_textField.setTextFormat(textFormat:TextFormat)  
  
textFormat のプロパティをテキストフィールドのすべてのテキストに適用します。
```

■ シンタックス 2:

```
my_textField.setTextFormat(beginIndex:Number, textFormat:TextFormat)  
  
textFormat のプロパティを beginIndex 位置の文字に適用します。
```

■ シンタックス 3:

```
my_textField.setTextFormat(beginIndex:Number, endIndex:Number,  
textFormat:TextFormat)  
  
textFormat パラメータのプロパティを beginIndex 位置から endIndex 位置までのテキスト  
範囲に適用します。
```

ユーザーによって手作業で挿入されたテキストは、テキストの挿入か所に指定されているフォーマットではなく、新しいテキスト用のテキストフィールドのデフォルトフォーマットが適用されます。新しいテキスト用のテキストフィールドのデフォルトフォーマットを設定するには、
TextField.setNewTextFormat() を使用します。

パラメータ

beginIndex:Number (オプション) - 該当するテキスト範囲の最初の文字を指定する整数。
beginIndex および endIndex を指定しない場合、TextFormat は TextField 全体に適用されます。

endIndex:Number (オプション) - 該当するテキスト範囲の直後の文字を指定する整数。
beginIndex を指定して endIndex を指定しない場合、TextFormat は beginIndex で指定された
1つの文字に適用されます。

textFormat:TextFormat - 文字と段落のフォーマット情報を含む TextFormat オブジェクト。

例

次の例では、テキストの 2 つの異なるストリングにテキストフォーマットを設定します。

setTextFormat() メソッドを呼び出し、my_txt テキストフィールドに適用します。

```
var format1_fmt:TextFormat = new TextFormat();  
format1_fmt.font = "Arial";  
var format2_fmt:TextFormat = new TextFormat();  
format2_fmt.font = "Courier";  
  
var string1:String = "Sample string number one."+newline;  
var string2:String = "Sample string number two."+newline;  
  
this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, 300, 200);  
my_txt.multiline = true;  
my_txt.wordWrap = true;  
my_txt.text = string1;  
var firstIndex:Number = my_txt.length;  
my_txt.text += string2;  
var secondIndex:Number = my_txt.length;  
  
my_txt.setTextFormat(0, firstIndex, format1_fmt);  
my_txt.setTextFormat(firstIndex, secondIndex, format2_fmt);
```

関連項目

[TextFormat, setNewTextFormat \(TextField.setNewTextFormat メソッド\)](#)

_soundbuftime (TextField._soundbuftime プロパティ)

public _soundbuftime : Number

サウンドのストリーミングを開始するまでにサウンドをプリバッファする秒数を指定します。

メモ: このプロパティを TextField オブジェクトに対して指定することができますが、実際には一口一
ドされたすべてのサウンドに適用されるグローバルプロパティであるので、単に _soundbuftime と
いう形で値を指定することもできます。このプロパティを TextField オブジェクトに対して設定する
と、実際にグローバルプロパティに設定されます。詳細および使用例については、_soundbuftime
を参照してください。

関連項目

[_soundbuftime プロパティ](#)

tabEnabled (TextField.tabEnabled プロパティ)

public tabEnabled : Boolean

テキストフィールドが自動タブ順に含まれるかどうかを指定します。デフォルト値は undefined です。

tabEnabled プロパティが undefined または true である場合、オブジェクトは自動タブ順に含ま
れます。tabIndex プロパティにも値を設定すると、オブジェクトはカスタムタブ順にも含まれます。
tabEnabled が false の場合は、tabIndex プロパティが設定されていても、オブジェクトは自動
タブ順またはカスタムタブ順に含まれません。

例

次の例では、複数のテキストフィールド one_txt、two_txt、three_txt、および four_txt を作
成します。テキストフィールド three_txt の tabEnabled プロパティは false に設定され、自動
タブ順から除外されます。

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100, 22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100, 22);
four_txt.border = true;
four_txt.type = "input";

three_txt.tabEnabled = false;
three_txt.text = "tabEnabled = false;";
```

関連項目

[tabEnabled \(Button.tabEnabled プロパティ\)](#), [tabEnabled \(MovieClip.tabEnabled プロパティ\)](#)

tabIndex (TextField.tabIndex プロパティ)

public tabIndex : Number

SWF ファイル内のオブジェクトのタブ順をカスタマイズできます。ボタン、ムービークリップ、またはテキストフィールドインスタンスの tabIndex プロパティを設定できます。デフォルトの値は undefined です。

SWF ファイルに現在表示されているオブジェクトに tabIndex プロパティがある場合は、自動タブ順が無効になり、SWF ファイルのオブジェクトの tabIndex プロパティからタブ順が計算されます。カスタムタブ順には、tabIndex プロパティを持つオブジェクトのみが含まれます。

tabIndex プロパティは正の整数である必要があります。オブジェクトのタブ順は、その tabIndex プロパティに従って昇順に決定されます。tabIndex の値が 1 であるオブジェクトは、tabIndex の値が 2 であるオブジェクトよりも優先されます。2 つのオブジェクトの tabIndex が同じ値である場合、オブジェクトのタブ順は undefined になります。

tabIndex プロパティで定義されるカスタムタブ順は flat です。つまり、SWF ファイル内のオブジェクトの階層関係は無視されます。SWF ファイルで tabIndex プロパティを持つすべてのオブジェクトは、タブ順に従って配置されます。タブ順は tabIndex の値の順番に従います。tabIndex の値が同じである 2 つのオブジェクト間では、優先順が undefined になります。複数のオブジェクトの tabIndex に同じ値を使用しないでください。

例

次の ActionScript は、4 つのテキストフィールドを動的に作成し、作成したテキストフィールドをカスタムタブ順に割り当てます。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100, 22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100, 22);
four_txt.border = true;
four_txt.type = "input";

one_txt.tabIndex = 3;
```

```
two_txt.tabIndex = 1;
three_txt.tabIndex = 2;
four_txt.tabIndex = 4;
```

関連項目

[tabIndex \(Button.tabIndex プロパティ\)](#), [tabIndex \(MovieClip.tabIndex プロパティ\)](#)

_target (TextField._target プロパティ)

public _target : [String](#) (読み取り専用)

テキストフィールドインスタンスのターゲットパス。`_self` は現在のウィンドウ内の現在のフレームを指定します。`_blank` は新しいウィンドウを指定します。`_parent` は現在のフレームの親を指定します。`_top` は現在のウィンドウ内のトップレベルのフレームを指定します。

例

次の ActionScript は、テキストフィールド `my_txt` を作成し、この新しいフィールドのターゲットパスをスラッシュ表記とドット表記の両方で出力します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
trace(my_txt._target); // output: /my_txt
trace(eval(my_txt._target)); // output: _level0.my_txt
```

text (TextField.text プロパティ)

public text : [String](#)

テキストフィールドの現在のテキストを示します。行はキャリッジリターン文字 ("r", ASCII 13) で区切られます。このプロパティは、テキストフィールドが HTML である場合でも、HTML タグが付いていない通常のフォーマットなしのテキストを示します。

例

次の例では、HTML テキストフィールド `my_txt` を作成し、HTML フォーマットのテキストストリングをこのフィールドに割り当てます。`htmlText` プロパティをトレースすると、HTML フォーマットのストリングが [出力] パネルに表示されます。HTML フォーマットのストリングがログファイルに書き込まれます。`text` プロパティの値をトレースすると、HTML タグの付いたフォーマットされていないストリングが [出力] パネルに表示されます。`text` プロパティの値をトレースすると、HTML タグの付いたフォーマットされていないストリングがログファイルに書き込まれます。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 400, 22);
my_txt.html = true;
my_txt.htmlText = "<b>Remember to always update the help panel.</b>";

trace("htmlText: "+my_txt.htmlText);
trace("text: "+my_txt.text);
```

```
// output:  
htmlText: <P ALIGN="LEFT"><FONT FACE="Times New Roman" SIZE="12"  
COLOR="#000000">  
<B>Remember to always update your help panel.</B></FONT></P>  
text: Remember to always update your help panel.
```

関連項目

[htmlText \(TextField.htmlText プロパティ\)](#)

textColor (TextField.textColor プロパティ)

public textColor : Number

テキストフィールドのテキストの色を示します。16進数カラーシステムでは、6桁の数値を使って色の値を示します。1つの桁で、16種類の値(文字)を指定できます。値の範囲は、0～9、A～Fです。黒は(#000000)と表され、その反対の白は(#FFFFFF)と表されます。

例

次の ActionScript を実行すると、テキストフィールドが作成され、そのカラー プロパティが赤に変更されます。

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);  
my_txt.text = "this will be red text";  
my_txt.textColor = 0xFF0000;
```

textHeight (TextField.textHeight プロパティ)

public textHeight : Number

テキストの高さを示します。

例

次の例では、テキストフィールドを作成し、そのフィールドにテキストストリングを割り当てます。trace ステートメントを使用して、テキストの高さと幅を [出力] パネルに表示します。trace() メソッドを使用して、テキストの高さと幅をログファイルに書き込みます。次に、autoSize プロパティを使用してテキストフィールドのサイズを変更し、新しい高さと幅も [出力] パネルに表示します。次に、autoSize プロパティを使用してテキストフィールドのサイズを変更し、新しい高さと幅もログファイルに書き込みます。

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);  
my_txt.text = "Sample text";  
trace("textHeight: "+my_txt.textHeight+, textWidth: "+my_txt.textWidth+");  
trace("_height: "+my_txt._height+, _width: "+my_txt._width+"\n");
```

```
my_txt.autoSize = true;
trace("after my_txt.autoSize = true;");
trace("_height: "+my_txt._height+", _width: "+my_txt._width);
次の情報が 출력されます。
textHeight: 15, textWidth: 56
_height: 300, _width: 100

after my_txt.autoSize = true;
_height: 19, _width: 60
```

関連項目

[textWidth \(TextField.textWidth プロパティ\)](#)

textWidth (TextField.textWidth プロパティ)

public textWidth : Number

テキストの幅を示します。

例

TextField.textHeight の例を参照してください。

関連項目

[textHeight \(TextField.textHeight プロパティ\)](#)

type (TextField.type プロパティ)

public type : String

テキストフィールドのタイプを指定します。2つの値があります。"dynamic" はダイナミックテキストフィールドを指定します。このフィールドをユーザーが編集することはできません。"input" はテキスト入力フィールドを指定します。

例

次の例では、2つのテキストフィールド username_txt および password_txt を作成します。両方のテキストフィールドにテキストが入力されますが、password_txt では password プロパティが true に設定されています。このため、password_txt に入力した文字はアスタリスクで表示されます。

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";
```

```
this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

_url (TextField._url プロパティ)

public _url : String (読み取り専用)

テキストフィールドを作成した SWF ファイルの URL を取得します。

例

次の例では、テキストフィールドを作成した SWF ファイルの URL と、ロードされた SWF ファイルを取得します。

```
this.createTextField("my_txt", 1, 10, 10, 100, 22);
trace(my_txt._url);
```

```
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._url);
};
var holder_mcl:MovieClipLoader = new MovieClipLoader();
holder_mcl.addListener(mclListener);
holder_mcl.loadClip("best_flash_ever.swf",
    this.createEmptyMovieClip("holder_mc", 2));
```

この例をテストすると、テストしている SWF ファイルの URL と "best_flash_ever.swf" ファイルが [出力] パネルに表示されます。この例をテストすると、テストしている SWF ファイルの URL と "best_flash_ever.swf" ファイルがログファイルに書き込まれます。

variable (TextField.variable プロパティ)

public variable : String

テキストフィールドに関連付けられた変数の名前です。このプロパティのタイプは String です。

例

次の例では、テキストフィールド my_txt を作成し、このフィールドに変数 today_date を関連付けています。変数 today_date を変更すると、my_txt に表示されるテキストが更新されます。

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.variable = "today_date";
var today_date:Date = new Date();

var date_interval:Number = setInterval(updateDate, 500);
```

```
function updateDate():Void {
    today_date = new Date();
}
```

_visible (TextField._visible プロパティ)

public _visible : Boolean

テキストフィールド my_txt が表示されるかどうかを示す布尔値です。_visible プロパティが false に設定されている非表示のテキストフィールドは、使用できません。

例

次の例では、テキストフィールド my_txt を作成します。ボタン visible_btn は、my_txt の表示と非表示を切り替えます。

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.background = true;
my_txt.backgroundColor = 0xDFDFDF;
my_txt.border = true;
my_txt.type = "input";

visible_btn.onRelease = function() {
    my_txt._visible = !my_txt._visible;
};
```

関連項目

[_visible \(Button._visible プロパティ\)](#), [_visible \(MovieClip._visible プロパティ\)](#)

_width (TextField._width プロパティ)

public _width : Number

ピクセル単位で示したテキストフィールドの幅。

例

次の例では、ステージ上の 3 番目のテキストフィールドの幅と高さを変更するときに使用できる 2 つのテキストフィールドを作成します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 40, 160, 120);
my_txt.background = true;
my_txt.backgroundColor = 0xFF0000;
my_txt.border = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.wordWrap = true;
```

```
this.createTextField("width_txt", this.getNextHighestDepth(), 10, 10, 30, 20);
width_txt.border = true;
width_txt.maxChars = 3;
width_txt.type = "input";
width_txt.text = my_txt._width;
width_txt.onChanged = function() {
    my_txt._width = this.text;
}

this.createTextField("height_txt", this.getNextHighestDepth(), 70, 10, 30, 20);
height_txt.border = true;
height_txt.maxChars = 3;
height_txt.type = "input";
height_txt.text = my_txt._height;
height_txt.onChanged = function() {
    my_txt._height = this.text;
}
```

この例をテストするときには、width_txt と height_txt に新しい値を入力して、my_txt のサイズを変更してみてください。

関連項目

[_height \(TextField._height プロパティ\)](#)

wordWrap (TextField.wordWrap プロパティ)

public wordWrap : Boolean

テキストフィールドのテキストを折り返すかどうかを示す布尔値。wordWrap の値が true である場合は、テキストフィールドのテキストを折り返し、false である場合は折り返しません。

例

次の例では、実行時に作成されるテキストフィールド内の長いテキストに、wordWrap の設定がどのように影響するかを示します。

```
this.createTextField("my_txt", 99, 10, 10, 100, 200);
my_txt.text = "This is very long text that will certainly extend beyond the width
    of this text field";
my_txt.border = true;
```

[制御]-[ムービープレビュー]を選択して、Flash Player で SWF ファイルをテストします。次に、ActionScript に戻ってコードに次の行を追加し、再度 SWF ファイルをテストします。

```
my_txt.wordWrap = true;
```

x (TextField._x プロパティ)

public _x : Number

親ムービークリップのローカル座標を基準にしてテキストフィールドの x 座標を設定する整数。テキストフィールドがメインのタイムラインにある場合、その座標系はステージの左上隅を (0,0) として参照します。変形されているムービークリップの内部にテキストフィールドがある場合、そのテキストフィールドの座標系はそれを囲むムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップに囲まれているテキストフィールドは、反時計回りに 90 度回転した座標系を継承します。テキストフィールドの座標は、基準点の位置を参照します。

例

次の例では、マウスをクリックした場所にテキストフィールドを作成します。作成したテキストフィールドには、そのテキストフィールドの現在の x 座標と y 座標が表示されます。

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 60, 22);
coords_txt.autoSize = true;
coords_txt.selectable = false;
coords_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    coords_txt.text = "X:"+Math.round(_xmouse)+", Y:"+Math.round(_ymouse);
    coords_txt._x = _xmouse;
    coords_txt._y = _ymouse;
};
Mouse.addListener(mouseListener);
```

関連項目

[_xscale \(TextField._xscale プロパティ\)](#), [_y \(TextField._y プロパティ\)](#),
[_yscale \(TextField._yscale プロパティ\)](#)

_xmouse (TextField._xmouse プロパティ)

public _xmouse : Number (読み取り専用)

テキストフィールドを基準にしたポインタの x 座標を返します。

メモ: このプロパティは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

次の例では、ステージ上に 3 つのテキストフィールドを作成します。mouse_txt インスタンスは、ステージを基準にしたマウスの現在位置を表示します。textfield_txt インスタンスは、my_txt インスタンスを基準としたマウスポインタの現在位置を表示します。次の ActionScript を FLA ファイルまたは ActionScript ファイルに追加します。

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 10, 10, 200, 22);
mouse_txt.border = true;
this.createTextField("textfield_txt", this.getNextHighestDepth(), 220, 10, 200,
    22);
textfield_txt.border = true;
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100, 160, 120);
my_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    mouse_txt.text = "MOUSE ... X:" + Math.round(_xmouse) + ",\tY:" +
    Math.round(_ymouse);
    textfield_txt.text = "TEXTFIELD ... X:" + Math.round(my_txt._xmouse) + ",\tY:" +
    Math.round(my_txt._ymouse);
}
}

Mouse.addListener(mouseListener);
```

関連項目

[_ymouse \(TextField._ymouse プロパティ \)](#)

_xscale (TextField._xscale プロパティ)

public _xscale : Number

テキストフィールドの基準点から適用する、テキストフィールドの水平方向の拡大 / 縮小率 (パーセンテージ) を決定します。デフォルトの基準点は (0,0) です。

例

次の例では、scaleUp_btn インスタンスおよび scaleDown_btn インスタンスをクリックしたときに、my_txt インスタンスを拡大 / 縮小します。

```
this.createTextField("my_txt", 99, 10, 40, 100, 22);
my_txt.autoSize = true;
my_txt.border = true;
my_txt.selectable = false;
my_txt.text = "Sample text goes here./";

scaleUp_btn.onRelease = function() {
    my_txt._xscale = 2;
```

```
    my_txt._yscale = 2;
}
scaleDown_btn.onRelease = function() {
    my_txt._xscale /= 2;
    my_txt._yscale /= 2;
}
```

関連項目

[_x \(TextField._x プロパティ\)](#), [_y \(TextField._y プロパティ\)](#),
[_yscale \(TextField._yscale プロパティ\)](#)

_y (TextField._y プロパティ)

public _y : Number

親ムービークリップのローカル座標を基準にしたテキストフィールドの y 座標。テキストフィールドがメインのタイムラインにある場合、その座標系はステージの左上隅を (0,0) として参照します。変形されているムービークリップの内部にテキストフィールドがある場合、そのテキストフィールドの座標系はそれを囲むムービークリップのローカル座標系になります。したがって、反時計回りに 90 度回転したムービークリップに囲まれているテキストフィールドは、反時計回りに 90 度回転した座標系を継承します。テキストフィールドの座標は、基準点の位置を参照します。

例

TextField._x の例を参照してください。

関連項目

[_x \(TextField._x プロパティ\)](#), [_xscale \(TextField._xscale プロパティ\)](#),
[_yscale \(TextField._yscale プロパティ\)](#)

_ymouse (TextField._ymouse プロパティ)

public _ymouse : Number (読み取り専用)

テキストフィールドを基準にしたポインタの y 座標を示します。

メモ: このプロパティは、System.capabilities.hasMouse が true である、または System.capabilities.hasStylus が true の場合のみ Flash Lite でサポートされます。

例

TextField._xmouse の例を参照してください。

関連項目

[_xmouse \(TextField._xmouse プロパティ\)](#)

`_yscale` (`TextField._yscale` プロパティ)

`public _yscale : Number`

テキストフィールドの基準点から適用する、テキストフィールドの垂直方向の拡大 / 縮小率 (パーセンテージ)。デフォルトの基準点は (0,0) です。

例

`TextField._xscale` の例を参照してください。

関連項目

[_x](#) (`TextField._x` プロパティ), [_xscale](#) (`TextField._xscale` プロパティ),
[_y](#) (`TextField._y` プロパティ)

TextFormat

`Object`

|
+-TextFormat

`public class TextFormat
extends Object`

`TextFormat` クラスは、文字フォーマット情報を表します。`TextFormat` クラスを使用して、テキストフィールドの特定のテキストフォーマットを作成します。静止テキストおよびダイナミックテキストフィールドの両方にテキストフォーマットを適用できます。`TextFormat` クラスの一部のプロパティは、埋め込みフォントとデバイスフォントのどちらにも使用できません。

関連項目

[setTextFormat](#) (`TextField.setTextFormat` メソッド),
[getTextFormat](#) (`TextField.getTextFormat` メソッド)

プロパティ一覧

オプション	プロパティ	説明
	<code>align:String</code>	段落の整列の設定を示すストリングです。
	<code>blockIndent:Number</code>	ブロックのインデントをポイント単位で示す数値。
	<code>bold:Boolean</code>	テキストフィールドがボールド体であるかどうかを指定するブール値。
	<code>bullet:Boolean</code>	テキストが箇条書きリストにあるかどうかを示すブール値。
	<code>color:Number</code>	テキストの色を示す数値。
	<code>font:String</code>	テキストのフォントの名前を指定するストリング。
	<code>indent:Number</code>	左マージンから段落の先頭文字までのインデントを示す整数。
	<code>italic:Boolean</code>	このテキストフォーマットのテキストをイタリックにするかどうかを示すブール値。
	<code>leading:Number</code>	行間の垂直の行送りを示す整数。
	<code>leftMargin:Number</code>	段落の左マージンをポイント単位で示します。
	<code>rightMargin:Number</code>	段落の右マージンをポイント単位で示します。
	<code>size:Number</code>	このテキストフォーマットでのテキストのポイントサイズ。
	<code>tabStops:Array</code>	カスタムタブストップを負以外の整数の配列として指定します。
	<code>target:String</code>	ハイパーリンクを表示するターゲットウィンドウを示します。
	<code>underline:Boolean</code>	このテキストフォーマットを使用するテキストにアンダーラインを表示するか(true)、または表示しないか(false)を示すブール値です。
	<code>url:String</code>	このテキストフォーマットを使用するテキストのハイパーリンク先の URL を示します。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

コンストラクター一覧

シグネチャ	説明
<code>TextFormat([font: String], [size:Number], [color:Number], [bold:Boolean], [italic:Boolean], [underline:Boolean], [url:String], [target:String], [align:String], [leftMargin:Number], [rightMargin:Number], [indent:Number], [leading:Number])</code>	指定されたプロパティを使用して TextFormat オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>getTextExtent (text:String, [width:Number]) : Object</code>	my_fmt に指定したフォーマットのテキストストリング text のテキスト寸法情報を返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

align (TextFormat.align プロパティ)

public align : String

段落の整列の設定を示すストリングです。静止テキストおよびダイナミックテキストにこのプロパティを適用できます。次の一覧に、このプロパティに指定できる値を示します。

- "left" は段落を左揃えにします。
- "center" は段落を中央揃えにします。
- "right" は段落を右揃えにします。

デフォルト値 null はプロパティを未設定にします。

例

次の例では、境界線のあるテキストフィールドを作成し、TextFormat.align を使用してテキストを中央揃えにします。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.align = "center";

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

blockIndent (TextFormat.blockIndent プロパティ)

public blockIndent : Number

ブロックのインデントをポイント単位で示す数値。ブロックのインデントは、テキストのブロック全体、つまりテキストのすべての行に適用されます。一方、通常のインデント (TextFormat.indent) は各段落の先頭行にのみ影響します。このプロパティが null である場合、TextFormat オブジェクトはブロックのインデントを指定しません。

例

この例では、境界線のあるテキストフィールドを作成し、ブロックのインデントを 20 に設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.blockIndent = 20;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

bold (TextFormat.bold プロパティ)

```
public bold : Boolean
```

テキストフィールドがボールド体であるかどうかを指定するブール値。デフォルト値 null はプロパティを未設定にします。値が true の場合は、テキストがボールド体になります。

メモ :アラビア語、ヘブライ語、タイ語の場合、このプロパティは段落レベルフォーマットでのみ機能します。

例

次の例では、ボールド体の文字を含むテキストフィールドを作成します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my text field object text";
my_txt.setTextFormat(my_fmt);
```

bullet (TextFormat.bullet プロパティ)

```
public bullet : Boolean
```

テキストが箇条書きリストにあるかどうかを示すブール値。箇条書きリストでは、テキストの各段落がインデントされます。箇条書きシンボルは、各段落の先頭行の左に表示されます。デフォルト値は null です。

メモ :Flash Lite では、このプロパティは埋め込みフォントに対してのみ使用できます。このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次の例では、実行時に新しいテキストフィールドを作成し、そのフィールドに改行を含むストリングを入力します。TextFormat クラスを使用して文字にフォーマットを設定し、テキストフィールドの各行に箇条書きシンボルを追加します。次の ActionScript を使用します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bullet = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my text"+newline;
my_txt.text += "this is more text"+newline;
my_txt.setTextFormat(my_fmt);
```

color (TextFormat.color プロパティ)

public color : Number

テキストの色を示す数値。たとえば、0xFF0000 は赤、0x00FF00 は緑など、3 つの 8 ビットの RGB コンポーネントを示す数値です。

メモ :アラビア語、ヘブライ語、タイ語の場合、このプロパティは段落レベルフォーマットでのみ機能します。

例

次の例では、テキストフィールドを作成し、テキストの色を赤に設定します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.blockIndent = 20;
my_fmt.color = 0xFF0000; // hex value for red

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

font (TextFormat.font プロパティ)

public font : String

テキストのフォントの名前を指定するストリング。デフォルト値 null はプロパティを未設定にします。

メモ :Flash Lite では、このプロパティは埋め込みフォントに対してのみ使用できます。このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次の例では、テキストフィールドを作成し、フォントを Courier に設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.font = "Courier";

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

getTextExtent (TextFormat.getTextExtent メソッド)

```
public getTextExtent(text:String, [width:Number]) : Object
```

my_fmt に指定したフォーマットのテキストストリング text のテキスト寸法情報を返します。text ストリングは標準テキスト（非 HTML）として扱われます。

このメソッドは、ascent、descent、width、height、textFieldHeight および textFieldWidth の 6 つのプロパティを持つオブジェクトを返します。すべてのプロパティ値はピクセル単位です。

width パラメータを指定した場合は、指定したテキストに折り返しが適用されます。これにより、指定したすべてのテキストを表示するために必要なテキストボックスの高さを判断することができます。

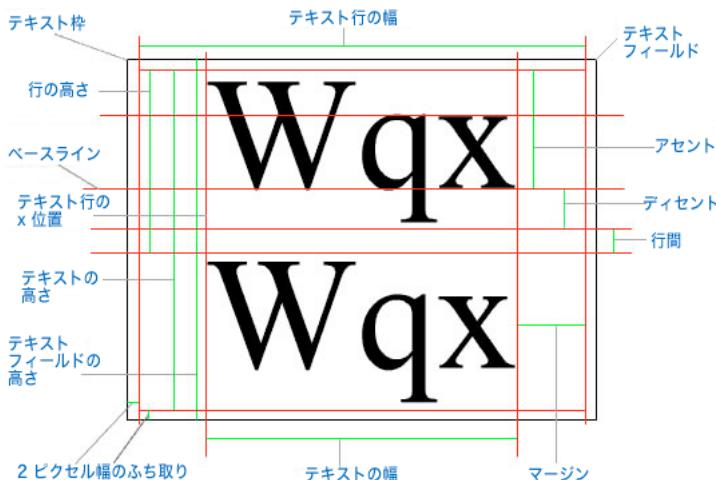
ascent プロパティはテキスト行のベースラインよりも上の長さ、descent プロパティはテキスト行のベースラインよりも下の長さを表します。最初のテキスト行のベースラインは、テキストフィールドの基準点に ascent パラメータ値を加算した位置になります。

width プロパティと height プロパティは、テキストストリングの幅と高さを示します。

textFieldHeight プロパティと textFieldWidth プロパティは、テキストストリング全体を表示するのに必要なテキストフィールドオブジェクトの高さと幅を表します。テキストフィールドの周囲には 2 ピクセル幅の " ふち取り " があるので、textFieldHeight の値は height+4 になります。同様に、textFieldWidth の値は常に width+4 になります。

テキストメトリクスに基づいてテキストフィールドを作成する場合は、height ではなく textFieldHeight を、width ではなく textFieldWidth を使用してください。

次の図に、この寸法を示します。



TextFormat オブジェクトをセットアップする際には、フォント名、フォントサイズ、行間を含め、テキストフィールドを作成する際とまったく同じようにすべての属性を設定する必要があります。行間のデフォルト値は 2 です。

パラメータ

`text:String` - ストリング。

`width:Number` (オプション) - 指定したテキストを折り返す幅を示す数値 (ピクセル単位)。

戻り値

`Object` - `width`、`height`、`ascent`、`descent`、`textFieldHeight`、`textFieldWidth` の各プロパティを持つオブジェクト。

例

次の例では、指定したフォーマットを使ったテキストストリングを表示する、ストリングと同じサイズの單一行テキストフィールドを作成します。

```
var my_str:String = "Small string";

// Create a TextFormat object,
// and apply its properties.
var my_fmt:TextFormat = new TextFormat();
with (my_fmt) {
    font = "Arial";
    bold = true;
}

// Obtain metrics information for the text string
// with the specified formatting.
var metrics:Object = my_fmt.getTextExtent(my_str);

// Create a text field just large enough to display the text.
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100,
    metrics.textFieldWidth,
    metrics.textFieldHeight);
my_txt.border = true;
my_txt.wordWrap = true;
// Assign the same text string and TextFormat object to the my_txt object.
my_txt.text = my_str;
my_txt.setTextFormat(my_fmt);

次の例では、指定したフォーマットのテキストストリングを表示できるだけの高さで、複数行の 100 ピクセル幅のテキストフィールドを作成します。

// Create a TextFormat object.
var my_fmt:TextFormat = new TextFormat();
// Specify formatting properties for the TextFormat object:
my_fmt.font = "Arial";
```

```

my_fmt.bold = true;
my_fmt.leading = 4;

// The string of text to be displayed
var textToDisplay:String = "Macromedia Flash Player 7, now with improved text
metrics./";

// Obtain text measurement information for the string,
// wrapped at 100 pixels.
var metrics:Object = my_fmt.getTextExtent(textToDisplay, 100);

// Create a new TextField object using the metric
// information just obtained.
this.createTextField("my_txt", this.getNextHighestDepth(), 50, 50-
    metrics.ascent, 100,
metrics.textFieldHeight);
my_txt.wordWrap = true;
my_txt.border = true;
// Assign the text and the TextFormat object to the TextObject:
my_txt.text = textToDisplay;
my_txt.setTextFormat(my_fmt);

```

indent (TextFormat.indent プロパティ)

`public indent : Number`

左マージンから段落の先頭文字までのインデントを示す整数。デフォルト値 null はプロパティを未設定にします。

例

次の例では、テキストフィールドを作成し、インデントを 10 に設定します。

```

this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.indent = 10;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);

```

関連項目

[blockIndent \(TextFormat.blockIndent プロパティ \)](#)

italic (TextFormat italic プロパティ)

public italic : Boolean

このテキストフォーマットのテキストをイタリックにするかどうかを示す布尔値。デフォルト値 null はプロパティを未設定にします。

メモ: アラビア語、ヘブライ語、タイ語の場合、このプロパティは段落レベルフォーマットでのみ機能します。

例

次の例では、テキストフィールドを作成し、テキストスタイルをイタリックに設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.italic = true;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

leading (TextFormat leading プロパティ)

public leading : Number

行間の垂直の行送りを示す整数。デフォルト値 null はプロパティを未設定にします。

例

次の例では、テキストフィールドを作成し、行間を 10 に設定します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.leading = 10;

this.createTextField("my_txt", 1, 100, 100, 100, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

leftMargin (TextFormat.leftMargin プロパティ)

public leftMargin : Number

段落の左マージンをポイント単位で示します。デフォルト値 null はプロパティを未設定にします。

例

次の例では、テキストフィールドを作成し、左マージンを 20 ポイントに設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.leftMargin = 20;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

rightMargin (TextFormat.rightMargin プロパティ)

public rightMargin : Number

段落の右マージンをポイント単位で示します。デフォルト値 null はプロパティを未設定にします。

例

次の例では、テキストフィールドを作成し、右マージンを 20 ポイントに設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.rightMargin = 20;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

size (TextFormat.size プロパティ)

public size : Number

このテキストフォーマットでのテキストのポイントサイズ。デフォルト値 null はプロパティを未設定にします。

メモ :アラビア語、ヘブライ語、タイ語の場合、このプロパティは段落レベルフォーマットでのみ機能します。

例

次の例では、テキストフィールドを作成し、テキストサイズを 20 ポイントに設定します。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.size = 20;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

tabStops (TextFormat.tabStops プロパティ)

public tabStops : Array

カスタムタブストップを負以外の整数の配列として指定します。各タブストップはピクセル単位で指定します。カスタムタブストップを指定しないと (null)、タブストップはデフォルトの 4(平均文字幅)になります。

メモ :Flash Lite では、このプロパティは埋め込みフォントに対してのみ使用できます。このプロパティはアラビア語、ヘブライ語、タイ語ではサポートされていません。

例

次の例では、タブストップが 40 ピクセルごとのテキストフィールドと、タブストップが 75 ピクセルごとのテキストフィールドを作成します。

```
this.createTextField("mytext",1,100,100,400,100);
mytext.border = true;
var myformat:TextFormat = new TextFormat();
myformat.tabStops = [40,80,120,160];
mytext.text = "A\tB\tC\tD"; // \t is the tab stop character
mytext.setTextFormat(myformat);

this.createTextField("mytext2",2,100,220,400,100);
mytext2.border = true;
```

```
var myformat2:TextFormat = new TextFormat();
myformat2.tabStops = [75,150,225,300];
mytext2.text ="A\tB\tC\tD";
mytext2.setTextFormat(myformat2);
```

target (TextFormat.target プロパティ)

public target : [String](#)

ハイパーリンクを表示するターゲットウィンドウを示します。ターゲットウィンドウが空のストリングである場合、テキストはデフォルトのターゲットウィンドウ _self に表示されます。カスタム名を選択することも、次の 4 つの名前のいずれかを選択することもできます。_self は現在のウィンドウ内の現在のフレームを指定します。_blank は新しいウィンドウを指定します。_parent は現在のフレームの親を指定します。_top は現在のウィンドウ内のトップレベルのフレームを指定します。TextFormat.url プロパティが空のストリングまたは null の場合は、このプロパティを取得および設定することはできますが、プロパティには何の影響もありません。

例

次の例では、Macromedia Web サイトへのハイパーリンクを持つテキストフィールドを作成します。この例では、TextFormat.target を使用して、Macromedia Web サイトを新しいブラウザウィンドウに表示します。

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";
myformat.target = "_blank";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

関連項目

[url \(TextFormat.url プロパティ \)](#)

TextFormat コンストラクタ

```
public TextFormat([font:String], [size:Number], [color:Number], [bold:Boolean],  
[italic:Boolean], [underline:Boolean], [url:String], [target:String],  
[align:String], [leftMargin:Number], [rightMargin:Number], [indent:Number],  
[leading:Number])
```

指定されたプロパティを使用して TextFormat オブジェクトを作成します。この TextFormat オブジェクトのプロパティを変更して、テキストフィールドのフォーマットを変更できます。

null 値を設定したパラメータは未設定になります。すべてのパラメータはオプションです。省略したパラメータは null として扱われます。

パラメータ

font:String(オプション)- テキストのフォント名を示すストリング。

size:Number(オプション)- ポイントサイズを示す整数。

color:Number(オプション)- このテキストフォーマットを使用するテキストの色。たとえば、0xFF0000 は赤、0x00FF00 は緑など、3 つの 8 ビットの RGB コンポーネントを示す数値です。

bold:Boolean(オプション)- テキストがボールド体であるかどうかを示すブール値。

italic:Boolean(オプション)- テキストがイタリック体であるかどうかを示すブール値。

underline:Boolean(オプション)- テキストが下線付きであるかどうかを示すブール値。

url:String(オプション)- このテキストフォーマットのテキストのハイパーリンク先である URL。url が空のストリングである場合、テキストにはハイパーリンクがありません。

target:String(オプション)- ハイパーリンクを表示するターゲットウィンドウ。ターゲットウィンドウが空のストリングである場合、テキストはデフォルトのターゲットウィンドウ _self に表示されます。url パラメータに空のストリングまたは null 値を指定した場合は、このプロパティを取得または設定することはできますが、プロパティには何の影響もありません。

align:String(オプション)- 段落の整列の設定を示すストリング。"left" は段落を左揃えにします。"center" は段落を中央揃えにします。"right" は段落を右揃えにします。

leftMargin:Number(オプション)- 段落の左マージンをポイント単位で示します。

rightMargin:Number(オプション)- 段落の右マージンをポイント単位で示します。

indent:Number(オプション)- 左マージンから段落の先頭文字までのインデントを示す整数。

leading:Number(オプション)- 行間の垂直の行送りを示す数値。

例

次の例では、TextFormat オブジェクトを作成し、stats_txt テキストフィールドをフォーマットしてから、テキストを表示する新しいテキストフィールドを作成します。

```
// Define a TextFormat which is used to format the stats_txt text field.  
var my_fmt:TextFormat = new TextFormat();  
my_fmt.bold = true;  
my_fmt.font = "Arial";  
my_fmt.size = 12;  
my_fmt.color = 0xFF0000;  
// Create a text field to display the player's statistics.  
this.createTextField("stats_txt", 5000, 10, 0, 530, 22);  
// Apply the TextFormat to the text field.  
stats_txt.setNewTextFormat(my_fmt);  
stats_txt.selectable = false;  
stats_txt.text = "Lorem ipsum dolor sit amet...";  
  
www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "animations.fla"  
ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンの  
フォルダに移動してサンプルにアクセスします。
```

underline (TextFormat.underline プロパティ)

public underline : Boolean

このテキストフォーマットを使用するテキストにアンダーラインを表示するか(true)、または表示しないか(false)を示す布尔値です。これは、<U> タグによって設定されるアンダーラインと似ていますが、このタグの場合はデセンダントが正しくスキップされないので、本物のアンダーラインではありません。デフォルト値 null はプロパティを未設定にします。

メモ: アラビア語、ヘブライ語、タイ語の場合、このプロパティは段落レベルフォーマットでのみ機能します。

例

次の例では、テキストフィールドを作成し、テキストスタイルを下線に設定します。

```
this.createTextField("mytext",1,100,100,200,100);  
mytext.multiline = true;  
mytext.wordWrap = true;  
mytext.border = true;  
  
var myformat:TextFormat = new TextFormat();  
myformat.underline = true;  
mytext.text = "This is my first text field object text";  
mytext.setTextFormat(myformat);
```

url (TextFormat.url プロパティ)

```
public url : String
```

このテキストフォーマットを使用するテキストのハイパーリンク先の URL を示します。url プロパティが空のストリングである場合、テキストにはハイパーリンクがありません。デフォルト値 null はプロパティを未設定にします。

例

この例では、Macromedia Web サイトへのハイパーリンクが設定されたテキストフィールドを作成します。

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

Video

```
Object
 |
 +-Video
```

```
public class Video
extends Object
```

Video クラスを使用すると、SWF ファイルに埋め込まれたビデオコンテンツ、ホストデバイス上にローカルで保存されたビデオコンテンツ、またはリモートからストリーミングされるビデオコンテンツを表示することができます。

メモ : Flash Lite 2.0 プレーヤーと Flash Player 7 ではビデオの処理方法が異なります。次のような主な違いがあります。

- Flash Player 7 では、埋め込まれたビデオデータまたはストリーミングされるビデオデータを直接レンダリングします。Flash Lite 2.0 プレーヤーでは、ビデオデータを直接レンダリングする代わりに、データをモバイルデバイスに渡します。

- Flash Player 7 は、FLV 以外にも多くのビデオ形式をサポートしています。Flash Lite 2.0 は、次のようなビデオの再生をサポートします。SWF ファイルに埋め込まれたビデオ、ホストデバイス上の別のファイルに含まれるビデオ、およびネットワーク経由でリアルタイムでストリーミングされるビデオをサポートします。Flash Lite 2.0 プレーヤーは、特定のモバイルデバイスがサポートしているビデオ形式のみをサポートします。
- Flash Player 7 では、SWF ファイルにデータをバンドルすることも、Video オブジェクトを使用し、NetStream オブジェクトまたは Camera オブジェクトをビデオ情報のソースとして割り当てることでデータをストリーミングすることもできます。しかし、Flash Lite 2.0 プレーヤーは、NetStream オブジェクトおよび Camera オブジェクトをサポートしていません。代わりに、Flash Lite 2.0 では、Video という新しい種類のライブラリシンボルを使用して、ソースビデオデータを埋め込んだり、モバイルデバイス用にビデオをストリーミングしたりします。Flash Lite 2.0 は、NetStream オブジェクトをサポートしていないので、Video クラスのメソッドとプロパティを使用してビデオの再生を制御します。

モバイルデバイスには、小さなプロセッサの処理速度、メモリの制約、および独自のエンコーディング形式という要件があるために、Flash Lite 2.0 は、ビデオ情報を直接レンダリングできません。サポートされるビデオファイル形式は、モバイルデバイスの製造元によって異なります。サポートされるビデオ形式の詳細については、アプリケーションの展開先のハードウェアプラットフォームを確認してください。

Flash Lite 2.0 は、次の Flash Player 7 の機能をサポートしていません。

- Flash Media Server からのビデオデータのストリーミング
- ビデオの録画

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve
プロパティ)
```

イベント一覧

イベント	説明
<code>onStatus = function(infoObject: Object) {}</code>	デバイスにより呼び出され、ステータスまたはエラー状態を示すコールバックハンドラ。

メソッド一覧

オプション	シグネチャ	説明
	<code>close() : Void</code>	ビデオの再生を停止し、この Video オブジェクトに関連付けられているメモリを解放して、画面上のビデオ領域をクリアします。
	<code>pause() : Void</code>	ビデオの再生を停止し、現在のフレームを画面上にレンダリングしたままにします。
	<code>play() : Boolean</code>	このメソッドを呼び出すと、ビデオソースが開かれ、ビデオの再生が開始されます。
	<code>resume() : Void</code>	このメソッドを呼び出すと、ビデオの再生が再開されます。
	<code>stop() : Void</code>	ビデオの再生を停止し、現在のフレームを画面上にレンダリングしたままにします。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

close (Video.close メソッド)

public close() : Void

ビデオの再生を停止し、この Video オブジェクトに関連付けられているメモリを解放して、画面上のビデオ領域をクリアします。

例

次の例では、Video オブジェクト video1 で再生されているビデオを閉じます。

```
video1.close()
```

関連項目

```
play (Video.play メソッド), pause (Video.pause メソッド),  
resume (Video.resume メソッド)
```

onStatus (Video.onStatus ハンドラ)

```
onStatus = function(infoObject:Object) {}
```

デバイスにより呼び出され、ステータスまたはエラー状態を示すコールバックハンドラ。

パラメータ

infoObject:Object - 次の 2 つのプロパティを持つ infoObject パラメータ。

- code:String エラー状態またはステータスの説明 (デバイス固有)。
- level:Number エラーの場合は 0 、成功の場合は 0 以外の数値 (デバイス固有)。

例

次の例は、ステータスまたはエラー状態を表示する Video.onStatus() 関数の作成方法を示します。

```
var v:Video; // v is a Video object on the stage.  
v.onStatus = function(o:Object)  
{  
    if ( o.level )  
    {  
        trace( "Video Status Msg (" + o.level + "): " + o.code );  
    }  
    else  
    {  
        trace( "Video Status Error: " + o.code );  
    }  
}  
v.play("a.vid");
```

pause (Video.pause メソッド)

```
public pause():Void
```

ビデオの再生を停止し、現在のフレームを画面上にレンダリングしたままにします。その後で、 Video.resume() を呼び出すと、現在の位置から再生が再開されます。

例

次の例では、ユーザーが close_btn インスタンスをクリックしたときに、 Video オブジェクト my_video 内で再生されているビデオを停止します。

```
// video1 is the name of a Video object on Stage  
video1.pause()
```

関連項目

[play \(Video.play メソッド\)](#), [stop \(Video.stop メソッド\)](#), [resume \(Video.resume メソッド\)](#)

play (Video.play メソッド)

```
public play() : Boolean
```

このメソッドを呼び出すと、ビデオソースが開かれ、ビデオの再生が開始されます。

戻り値

[Boolean](#) - モバイルデバイスがビデオをレンダリングできる場合は true を返します。レンダリングできない場合は false を返します。

例

次の例では、Video オブジェクト video1 内で再生されている video1.flv を一時停止し、クリアします。

```
video1.play( "http://www.macromedia.com/samples/videos/clock.3gp" );
```

ステージ上の Video オブジェクトを使用して、バンドルされた複数のデバイスピデオをライブラリから直接再生することもできます。この場合は、アプリケーションのライブラリでデバイスピデオをバンドルします。また、ActionScript でビデオシンボルを参照できるようビデオシンボルに識別子を割り当てます。次の例に示すように、シンボルの ActionScript 識別子を Video.play() メソッドに渡して、ライブラリからデバイスピデオを再生できます。

```
placeHolderVideo.play("symbol://ocean_video");
```

ライブラリからビデオを再生する方法の詳細については、『Flash Lite 2.x アプリケーションの開発』の「ライブラリからバンドルされたビデオを直接再生」を参照してください。

関連項目

[stop \(Video.stop メソッド\)](#), [pause \(Video.pause メソッド\)](#),
[resume \(Video.resume メソッド\)](#)

resume (Video.resume メソッド)

```
public resume() : Void
```

このメソッドを呼び出すと、ビデオの再生が再開されます。

Video.pause() がその前に呼び出されていた場合は、現在の位置から再生が開始されます。

Video.stop() がその前に呼び出されていた場合は、最初のフレームから再生が開始されます。

例

次の例では、Video オブジェクト video1 内で再生されているビデオを再開します。

```
video1.resume()
```

関連項目

[pause \(Video.pause メソッド\)](#), [stop \(Video.stop メソッド\)](#)

stop (Video.stop メソッド)

```
public stop() : Void
```

ビデオの再生を停止し、現在のフレームを画面上にレンダリングしたままにします。その後で、`Video.resume()` を呼び出すと、ビデオの最初のフレームから再生が再開されます。

例

次の例では、ユーザーが `close_btn` インスタンスをクリックしたときに、`Video` オブジェクト `my_video` 内で再生されているビデオを停止します。

```
// video1 is the name of a Video object on Stage  
video1.stop();
```

関連項目

[play \(Video.play メソッド\)](#), [pause \(Video.pause メソッド\)](#),
[resume \(Video.resume メソッド\)](#)

XML

```
Object  
|  
+- XMLNode  
|  
+- XML
```

```
public class XML  
extends XMLNode
```

`XML` ドキュメントツリーをロード、解析、送信、構築、および操作するには、`XML` クラスのメソッドとプロパティを使用します。

`XML` クラスのメソッドを呼び出す前に、コンストラクタ `new XML()` を使用して `XML` オブジェクトのインスタンスを作成する必要があります。

`XML` ドキュメントは、Flash では `XML` クラスで表現されます。階層ドキュメントの各エレメントが、`XMLNode` オブジェクトで表されます。

次のメソッドとプロパティの詳細については、`XMLNode` クラスを参照してください。

`appendChild()`, `attributes`, `childNodes`, `cloneNode()`, `firstChild`, `hasChildNodes()`,
`insertBefore()`, `lastChild`, `nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `parentNode`,
`previousSibling`, `removeNode()`, `toString()`。

『ActionScript リファレンスガイド』の前のバージョンでは、上記のメソッドとプロパティは、`XML` クラスで説明されていました。現在は `XMLNode` クラスで説明されています。

メモ: XML オブジェクトおよび XMLNode オブジェクトは、W3C DOM レベル1勧告 (<http://www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html>) に準拠しています。この勧告では、Node インターフェイスおよび Document インターフェイスが規定されています。Document インターフェイスは Node インターフェイスから継承され、createElement() や createTextNode() などのメソッドが追加されています。ActionScript では、XML オブジェクトと XMLNode オブジェクトの機能の違いは、同様の方針で設計されています。

関連項目

[appendChild \(XMLNode.appendChild メソッド\)](#), [attributes \(XMLNode.attributes プロパティ\)](#), [childNodes \(XMLNode.childNodes プロパティ\)](#), [cloneNode \(XMLNode.cloneNode メソッド\)](#), [firstChild \(XMLNode.firstChild プロパティ\)](#), [hasChildNodes \(XMLNode.hasChildNodes メソッド\)](#), [insertBefore \(XMLNode.insertBefore メソッド\)](#), [lastChild \(XMLNode.lastChild プロパティ\)](#), [nextSibling \(XMLNode.nextSibling プロパティ\)](#), [nodeName \(XMLNode.nodeName プロパティ\)](#), [nodeType \(XMLNode.nodeType プロパティ\)](#), [nodeValue \(XMLNode.nodeValue プロパティ\)](#), [parentNode \(XMLNode.parentNode プロパティ\)](#), [previousSibling \(XMLNode.previousSibling プロパティ\)](#), [removeNode \(XMLNode.removeNode メソッド\)](#), [toString \(XMLNode.toString メソッド\)](#)

プロパティ一覧

オプション	プロパティ	説明
	<code>contentType:String</code>	XML.send() メソッドまたは XML.sendAndLoad() メソッドを呼び出したときにサーバーに送られる MIME コンテンツタイプ。
	<code>docTypeDecl:String</code>	XML ドキュメントの DOCTYPE 宣言についての情報を指定します。
	<code>ignoreWhite:Boolean</code>	デフォルト設定は <code>false</code> です。
	<code>loaded:Boolean</code>	XML ドキュメントが正常にロードされたかどうかを示します。
	<code>status:Number</code>	XML ドキュメントが XML オブジェクトに正常に解析されたかどうかを示す数値を自動的に設定し、返します。
	<code>xmlDecl:String</code>	ドキュメントの XML 宣言についての情報を指定するストリング。

XMLNode クラスから継承されるプロパティ

```
attributes (XMLNode.attributes プロパティ), childNodes (XMLNode.childNodes プロパティ), firstChild (XMLNode.firstChild プロパティ),  
lastChild (XMLNode.lastChild プロパティ), nextSibling (XMLNode.nextSibling プロパティ), nodeName (XMLNode.nodeName プロパティ), nodeType (XMLNode.nodeType プロパティ), nodeValue (XMLNode.nodeValue プロパティ),  
parentNode (XMLNode.parentNode プロパティ), previousSibling (XMLNode.previousSibling プロパティ)
```

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロパティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve プロパティ)
```

イベント一覧

イベント	説明
onData = function(src:String) {}	サーバーからの XML テキストのダウンロードが完了するか、サーバーからの XML テキストのダウンロード中にエラーが発生すると呼び出されます。
onLoad = function(success:Boolean) {}	XML ドキュメントをサーバーから受信すると、Flash Player によって呼び出されます。

コンストラクター一覧

シグネチャ	説明
XML(text:String)	新しい XML オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	addRequestHeader (header:Object, headerValue:String) : Void	POST アクションによって送信される HTTP リクエスト ヘッダ(Content-Type や SOAPAction など)を追加または変更します。
	createElement(name: String) : XMLNode	パラメータで指定された名前を持つ新しい XML エレメントを作成します。

オプション	シグネチャ	説明
	<code>createTextNode (value:String) : XMLNode</code>	指定されたテキストを持つ新しい XML テキストノードを作成します。
	<code>getBytesLoaded() : Number</code>	XML ドキュメントに対してロード(ストリーミング)されたバイト数を返します。
	<code>getBytesTotal() : Number</code>	XML ドキュメントのサイズをバイト単位で返します。
	<code>load(url:String) : Boolean</code>	指定された URL から XML ドキュメントを読み込み、指定された XML オブジェクトの内容をダウンロードされた XML データで置き換えます。
	<code>parseXML (value:String) : Void</code>	value パラメータで指定された XML テキストを解析し、指定された XML オブジェクトに XML ツリーを設定します。
	<code>send(url:String, [target:String], method:String) : Boolean</code>	指定された XML オブジェクトを XML ドキュメントにエンコードし、ブラウザの POST メソッドを使用して指定された URL に送ります。
	<code>sendAndLoad (url:String, resultXML:XML) : Void</code>	指定された XML オブジェクトを XML ドキュメントにエンコードし、POST メソッドを使用して、指定された URL に送ります。さらに、サーバーの応答をダウンロードし、パラメータで指定された resultXMLObject にその応答をロードします。

XMLNode クラスから継承されるメソッド

```
appendChild (XMLNode.appendChild メソッド), cloneNode (XMLNode.cloneNode メソッド), hasChildNodes (XMLNode.hasChildNodes メソッド), insertBefore (XMLNode.insertBefore メソッド), removeNode (XMLNode.removeNode メソッド), toString (XMLNode.toString メソッド)
```

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty (Object.hasOwnProperty メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), isPrototypeOf (Object.isPrototypeOf メソッド), registerClass (Object.registerClass メソッド), toString (Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf (Object.valueOf メソッド), watch (Object.watch メソッド)
```

addRequestHeader (XML.addRequestHeader メソッド)

```
public addRequestHeader(header:Object, headerValue:String) : Void
```

POST アクションによって送信される HTTP リクエストヘッダ (Content-Type や SOAPAction など) を追加または変更します。シンタックス 1 では、header および headerValue の 2 つのストリングをメソッドに渡します。シンタックス 2 では、ヘッダー名とヘッダー値を交互に含むストリングの配列を渡します。

同じヘッダー名に対して複数の呼び出しを実行すると、呼び出しのたびに前の呼び出しで設定された値が上書きされます。

このメソッドを使用して、標準の HTTP ヘッダー (Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning および WWW-Authenticate) を追加または変更することはできません。

パラメータ

header:Object - HTTP リクエストヘッダー名を表すストリング。

headerValue:String - header に関連付けられた値を表すストリング。

例

次の例では、SOAPAction というカスタム HTTP ヘッダーを my_xml という XML オブジェクトに追加します。値は Foo です。

```
my_xml.addRequestHeader("SOAPAction", "'Foo');
```

次の例では、headers という配列を作成します。この配列には、HTTP ヘッダーとその値が交互に格納されます。この配列を addRequestHeader() メソッドにパラメータとして渡します。

```
var headers:Array = new Array("Content-Type", "text/plain",
  "X-ClientAppVersion", "2.0");
my_xml.addRequestHeader(headers);
```

関連項目

[addRequestHeader \(LoadVars.addRequestHeader メソッド\)](#)

contentType (XML.contentType プロパティ)

public contentType : String

XML.send() メソッドまたは XML.sendAndLoad() メソッドを呼び出したときにサーバーに送られる MIME コンテンツタイプ。デフォルト値は application/x-www-form-urlencoded です。これはほとんどの HTML フォームで使用される標準の MIME コンテンツタイプです。

例

次の例では、新しい XML ドキュメントを作成し、そのドキュメントのデフォルトのコンテンツタイプを確認します。

```
// create a new XML document  
var doc:XML = new XML();
```

```
// trace the default content type  
trace(doc.contentType); // output: application/x-www-form-urlencoded
```

次の例では、XML パケットを定義し、XML オブジェクトのコンテンツタイプを設定します。その後データをサーバーに送り、ブラウザウィンドウに結果を表示します。

```
var my_xml:XML = new XML("<highscore><name>Ernie</name><score>13045</score></highscore>");  
my_xml.contentType = "text/xml";  
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

F12 キーを押して、ブラウザ内でこの例をテストします。

関連項目

[send \(XML.send メソッド\)](#), [sendAndLoad \(XML.sendAndLoad メソッド\)](#)

createElement (XML.createElement メソッド)

public createElement(name:String) : XMLNode

パラメータで指定された名前を持つ新しい XML エレメントを作成します。初期状態では、新しいエレメントには、親、子、および兄弟はありません。このメソッドは、エレメントとして新しく作成された XML オブジェクトへの参照を返します。このメソッドと XML.createTextNode() メソッドは、XML オブジェクトのノードを作成するためのコンストラクタメソッドです。

パラメータ

name:String - 作成する XML エレメントのタグ名。

戻り値

XMLNode - XMLNode オブジェクト。XML エレメントです。

例

次の例では、createElement() メソッドを使用して 3 つの XML ノードを作成します。

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

trace(doc);
// output: <element1><element2 /><element3 /></element1>
```

関連項目

[createTextNode \(XML.createTextNode メソッド\)](#)

createTextNode (XML.createTextNode メソッド)

`public createTextNode(value:String) : XMLNode`

指定されたテキストを持つ新しい XML テキストノードを作成します。初めは、新しいノードには親がありません。また、テキストノードは、子または兄弟を持つことができません。このメソッドは、新しいテキストノードを表す XML オブジェクトへの参照を返します。このメソッドと XML.createElement() メソッドは、XML オブジェクトのノードを作成するためのコンストラクタメソッドです。

パラメータ

`value:String` - ストリング。新しいテキストノードを作成するためのテキストです。

戻り値

`XMLNode` - XMLNode オブジェクト。

例

次の例では、createTextNode() メソッドを使用して XML テキストノードを 2 つ作成し、それらを既存の XML ノードに追加します。

```
// create an XML document
var doc:XML = new XML();
```

```

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

// create two XML text nodes using createTextNode()
var textNode1:XMLNode = doc.createTextNode("textNode1 String value");
var textNode2:XMLNode = doc.createTextNode("textNode2 String value");

// place the new nodes into the XML tree
element2.appendChild(textNode1);
element3.appendChild(textNode2);

trace(doc);
// output (with line breaks added between tags):
// <element1>
// <element2>textNode1 String value</element2>
// <element3>textNode2 String value</element3>
// </element1>

```

関連項目

[createElement \(XML.createElement メソッド\)](#)

docTypeDecl (XML.docTypeDecl プロパティ)

public docTypeDecl : [String](#)

XML ドキュメントの DOCTYPE 宣言についての情報を指定します。XML テキストが XML オブジェクトに解析された後、XML オブジェクトの XML.docTypeDecl プロパティは、XML ドキュメントの DOCTYPE 宣言のテキスト (たとえば、`<!DOCTYPE greeting SYSTEM "hello.dtd">`) に設定されます。このプロパティは、XML ノードオブジェクトでなく DOCTYPE 宣言のストリング表現を使用して設定されます。

ActionScript の XML パーサーは、妥当性検証用パーサーではありません。DOCTYPE 宣言はパーサーにより読み取られ、XML.docTypeDecl プロパティに格納されますが、DTD の妥当性検査は行われません。

解析中に DOCTYPE 宣言が見つからなかった場合、XML.docTypeDecl プロパティは `undefined` に設定されます。XML.toString() メソッドは、XML.xmlDecl に保存されている XML 宣言のすぐ後、XML オブジェクト内の他のテキストよりも前に、XML.docTypeDecl の内容を出力します。XML.docTypeDecl が `undefined` である場合、DOCTYPE 宣言は出力されません。

例

次の例では、`XML.docTypeDecl` プロパティを使用して、`XML` オブジェクトに DOCTYPE 宣言を設定します。

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

関連項目

[xmlDecl \(XML.xmlDecl プロパティ\)](#)

getBytesLoaded (XML.getBytesLoaded メソッド)

`public getBytesLoaded(): Number`

`XML` ドキュメントに対してロード(ストリーミング)されたバイト数を返します。`getBytesLoaded()` の値と `getBytesTotal()` の値を比較すると、`XML` ドキュメントの何 % がロードされたかを判断できます。

戻り値

`Number` - ロードされたバイト数を示す整数。

例

次の例では、`XML.getBytesLoaded()` メソッドと `XML.getBytesTotal()` メソッドを使用して `XML.load()` コマンドの進捗状況を出力する方法を示します。`XML.load()` コマンドの URL パラメータは、HTTP を使用する有効な XML ファイルを指定するように値を置き換える必要があります。この例を使用してハードディスク上にあるローカルファイルをロードしようとしても、ムービープレビューモードの Flash Player ではローカルファイル全体がロードされるため、正常に機能しません。

```
// create a new XML document
var doc:XML = new XML();

var checkProgress = function(xmlObj:XML) {
    var bytesLoaded:Number = xmlObj.getBytesLoaded();
    var bytesTotal:Number = xmlObj.getBytesTotal();
    var percentLoaded:Number = Math.floor((bytesLoaded / bytesTotal) * 100);
    trace ("milliseconds elapsed: " + getTimer());
    trace ("bytesLoaded: " + bytesLoaded);
    trace ("bytesTotal: " + bytesTotal);
    trace ("percent loaded: " + percentLoaded);
    trace ("-----");
}

doc.onLoad = function(success:Boolean) {
    clearInterval(intervalID);
    trace("intervalID: " + intervalID);
}
```

```
doc.load("[place a valid URL pointing to an XML file here]");
var intervalID:Number = setInterval(checkProgress, 100, doc);
```

関連項目

[getBytesTotal \(XML.getBytesTotal メソッド\)](#)

getBytesTotal (XML.getBytesTotal メソッド)

public getBytesTotal() : Number

XML ドキュメントのサイズをバイト単位で返します。

戻り値

Number - 整数。

例

`XML.getBytesLoaded()` の例を参照してください。

関連項目

[getBytesLoaded \(XML.getBytesLoaded メソッド\)](#)

ignoreWhite (XML.ignoreWhite プロパティ)

public ignoreWhite : Boolean

デフォルト設定は `false` です。`true` を設定すると、空白のみを含むテキストノードは解析処理中に破棄されます。先行空白または後続空白があるテキストノードは影響を受けません。

シンタックス1: 次のコードに示すように、XML オブジェクトごとに `ignoreWhite` プロパティを設定できます。

```
my_xml.ignoreWhite = true;
```

シンタックス2: 次のコードに示すように、XML オブジェクトにデフォルトの `ignoreWhite` プロパティを設定できます。

```
XML.prototype.ignoreWhite = true;
```

例

次の例では、空白のみのテキストノードが含まれる XML ファイルをロードします。`foyer` タグは 14 個の空白文字で構成されます。この例を実行するには、"flooring.xml" という名前のテキストファイルを作成し、そのファイルに次のタグをコピーします。

```
<house>
  <kitchen> ceramic tile </kitchen>
  <bathroom>linoleum</bathroom>
```

```

<foyer> </foyer>
</house>

"flooring.fla" という名前で新しい Flash ドキュメントを作成し、それを XML ファイルと同じディレクトリに保存します。次のコードをメインタイムラインに配置します。

// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success:Boolean) {
    trace(flooring);
}

// load the XML into the flooring object
flooring.load("flooring.xml");

// output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer />
</house>

```

`flooring.ignoreWhite` の設定を `false` に変更するか、単純に `flooring.ignoreWhite` が含まれるコード行を削除する場合、`foyer` タグの 14 個の空白文字は削除されません。

```

...
// set the ignoreWhite property to false (default value)
flooring.ignoreWhite = false;
...
// output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer> </foyer>
</house>

```

www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "XML_blogTracker.fla" および "XML_languagePicker.fla" ファイルにも例があります。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

load (XML.load メソッド)

```
public load(url:String) : Boolean
```

指定された URL から XML ドキュメントを読み込み、指定された XML オブジェクトの内容をダウンロードされた XML データで置き換えます。URL は相対 URL で、HTTP を使用して呼び出されます。ロード処理は非同期です。このため、load() メソッドの実行が完了しても、すぐには処理は終了しません。

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、url パラメータは、呼び出し元の SWF ファイルと同じスープードメインに属している必要があります。スープードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、www.someDomain.com に存在する SWF ファイルは、store.someDomain.com に存在するソースからデータをロードできます。これは、どちらのファイルも同じスープードメイン someDomain.com に属しているためです。

Flash Player 7 以降で SWF ファイルを実行している場合、url パラメータは正確に同じドメインに置かれている必要があります。たとえば www.someDomain.com に置かれている SWF ファイルは、www.someDomain.com に置かれているソースからのみデータをロードできます。異なるドメインからデータをロードする場合は、SWF ファイルをホストするサーバーに "クロスドメインポリシー ファイル" を置いておく必要があります。

load() メソッドを実行すると、XML オブジェクトの loaded プロパティが false に設定されます。XML データのダウンロードが終了すると、loaded プロパティが true に設定され、onLoad イベントハンドラが呼び出されます。XML データは、完全にダウンロードされるまで解析されません。XML オブジェクトに XML ツリーが既に含まれていた場合、その XML ツリーは破棄されます。

カスタム関数を定義して、XML オブジェクトの onLoad イベントハンドラが呼び出されたときに実行することができます。

パラメータ

url:String - ロードする XML ドキュメントが置かれている場所の URL を表すストリング。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、url は SWF ファイルと同じドメインに属している必要があります。詳細については、「説明」を参照してください。

戻り値

Boolean - パラメータが渡されなかった場合(パラメータが null の場合)は false、それ以外の場合は true を返します。onLoad() イベントハンドラを使用して、XML ドキュメントが正常にロードされたかどうかをチェックします。

例

次の例では、簡単な例では XML.load() メソッドを使用しています。

```
// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success) {
    trace(flooring);
};

// load the XML into the flooring object
flooring.load("flooring.xml");

"flooring.xml" ファイルの内容と、この例で生成される結果については、XML.ignoreWhite の例を
参照してください。
```

関連項目

[ignoreWhite \(XML.ignoreWhite プロパティ\)](#), [loaded \(XML.loaded プロパティ\)](#), [onLoad \(XML.onLoad ハンドラ\)](#)

loaded (XML.loaded プロパティ)

public loaded : Boolean

XML ドキュメントが正常にロードされたかどうかを示します。XML オブジェクト用に onLoad() イベントハンドラが定義されていない場合、XML.load() の呼び出しで開始されたドキュメントのロード処理が正常に完了した場合は true になり、それ以外の場合は false になります。ただし、XML オブジェクトの onLoad() イベントハンドラのカスタムビヘイビアを定義した場合は、必ずその関数で onload を設定してください。

例

次の例では、簡単なスクリプトで XML.loaded プロパティを使用します。

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    trace("success: "+success);
    trace("loaded: "+my_xml.loaded);
    trace("status: "+my_xml.status);
};
my_xml.load("http://www.flash-mx.com/mm/problems/products.xml");
```

`onLoad` ハンドラが呼び出されると、[出力] パネルに情報が表示されます。呼び出しが正常に終了すると、[出力] パネルに `loaded` ステータスが `true` と表示されます。

```
success: true  
loaded: true  
status: 0
```

関連項目

[load \(XML.load メソッド\)](#), [onLoad \(XML.onLoad ハンドラ\)](#)

onData (XML.onData ハンドラ)

```
onData = function(src:String) {}
```

サーバーからの XML テキストのダウンロードが完了するか、サーバーからの XML テキストのダウンロード中にエラーが発生すると呼び出されます。このハンドラは XML の解析前に呼び出されるため、このハンドラを使用して、Flash XML パーサーを使用する代わりにカスタム解析ルーチンを呼び出すことができます。`src` パラメータは、ダウンロード中にエラーが発生しなければ、サーバーからダウンロードされた XML テキストを含むストリングになります。エラーが発生した場合、`src` パラメータは `undefined` になります。

デフォルトでは、`XML.onData` イベントハンドラは `XML.onLoad` を呼び出します。`XML.onData` イベントハンドラを無効にしてカスタムビヘイビアを使用することができます。ただし、その場合、自分で実装する `XML.onData` で呼び出さない限り、`XML.onLoad` は呼び出されません。

パラメータ

`src:String` - スtring または `undefined`。サーバーから送られる生の(通常は XML 形式の)データです。

例

次の例では、`XML.onData` イベントハンドラのデフォルトの動作を示します。

```
XML.prototype.onData = function (src:String) {  
    if (src == undefined) {  
        this.onLoad(false);  
    } else {  
        this.parseXML(src);  
        this.loaded = true;  
        this.onLoad(true);  
    }  
}
```

`XML.onData` イベントハンドラをオーバーライド(上書き定義)して、XML テキストを解析せずに取得することができます。

関連項目

[onLoad \(XML.onLoad ハンドラ\)](#)

onLoad (XML.onLoad ハンドラ)

```
onLoad = function(success:Boolean) {}
```

XML ドキュメントをサーバーから受信すると、Flash Player によって呼び出されます。XML ドキュメントが正常に受信された場合、success パラメータは true です。ファイルが受信されなかつたか、サーバーから応答を受信する際にエラーが発生した場合は、success パラメータには false が渡されます。デフォルトでは、このメソッドの実行形態は非アクティブになります。デフォルトの実行形態を無効にするには、カスタムアクションを含む関数を割り当てます。

パラメータ

success:Boolean - XML.load() または XML.sendAndLoad() の処理によって XML オブジェクトが正常にロードされた場合は true、それ以外の場合は false になるブール値。

例

次の例には、簡単な電子商取引店頭アプリケーション用の ActionScript が含まれています。

sendAndLoad() メソッドは、ユーザーの名前とパスワードを含む XML エレメントを転送し、XML.onLoad ハンドラを使用してサーバーからの応答を処理します。

```
var login_str:String = "<login username=\""+username_txt.text+"\""
    password=\""+password_txt.text+"\\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();

myLoginReply_xml.ignoreWhite = true;

myLoginReply_xml.onLoad = function(success:Boolean){

    if (success) {

        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }

    } else {
        gotoAndStop("connectionFailed");
    }

};

my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",
    myLoginReply_xml);
```

関連項目

[load \(XML.load メソッド\)](#), [sendAndLoad \(XML.sendAndLoad メソッド\)](#)

parseXML (XML.parseXML メソッド)

```
public parseXML(value:String) : Void
```

value パラメータで指定された XML テキストを解析し、指定された XML オブジェクトに XML ツリーを設定します。XML オブジェクト内の既存のツリーはすべて破棄されます。

パラメータ

value:String - 解析後に指定の XML オブジェクトに渡される XML テキストを表すストリング。

例

次の例では、XML パケットを作成して解析します。

```
var xml_str:String = "<state name=\"California\">
<city>San Francisco</city></state>"

// defining the XML source within the XML constructor:
var my1_xml:XML = new XML(xml_str);
trace(my1_xml.firstChild.attributes.name); // output: California

// defining the XML source using the XML.parseXML method:
var my2_xml:XML = new XML();
my2_xml.parseXML(xml_str);
trace(my2_xml.firstChild.attributes.name); // output: California
```

send (XML.send メソッド)

```
public send(url:String, [target:String], method:String) : Boolean
```

指定された XML オブジェクトを XML ドキュメントにエンコードし、ブラウザの POST メソッドを使用して指定された URL に送ります。Flash のテスト環境では、GET メソッドのみを使用します。

パラメータ

url:String - ストリング。指定された XML オブジェクトの宛先 URL です。

target:String (オプション)- ストリング。サーバーが返したデータを表示するブラウザウィンドウ。

- _self は、現在のウィンドウ内の現在のフレームを指定します。
- _blank は、新規ウィンドウを指定します。
- _parent は、現在のフレームの親を指定します。
- _top は、現在のウィンドウ内の最上位のフレームを指定します。
window パラメータを指定しない場合は、_self を指定したのと同じ結果になります。

method:String -

戻り値

Boolean -

例

次の例では、XML パケットを定義し、XML オブジェクトのコンテンツタイプを設定します。その後データをサーバーに送り、ブラウザウィンドウに結果を表示します。

```
var my_xml:XML = new XML("<highscore><name>Ernie</name>
<score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

F12 キーを押して、ブラウザ内でこの例をテストします。

関連項目

[sendAndLoad \(XML.sendAndLoad メソッド\)](#)

sendAndLoad (XML.sendAndLoad メソッド)

public sendAndLoad(url:String, resultXML:XML) : Void

指定された XML オブジェクトを XML ドキュメントにエンコードし、POST メソッドを使用して、指定された URL に送ります。さらに、サーバーの応答をダウンロードし、パラメータで指定された resultXMLObject にその応答をロードします。サーバーの応答は、XML.load() メソッドで使用される方法と同じ方法でロードされます。

Flash Player 7 より前のバージョンの Player で SWF ファイルを実行している場合、url パラメータは、呼び出し元の SWF ファイルと同じスパーードメインに属している必要があります。スパーードメインは、ファイルの URL の左端の要素を削除することで求められます。たとえば、www.someDomain.com に存在する SWF ファイルは、store.someDomain.com に存在するソースからデータをロードできます。これは、どちらのファイルも同じスパーードメイン someDomain.com に属しているためです。

Flash Player 7 以降で SWF ファイルを実行している場合、url パラメータは正確に同じドメインに置かれている必要があります。たとえば www.someDomain.com に置かれている SWF ファイルは、www.someDomain.com に置かれているソースからのみデータをロードできます。異なるドメインからデータをロードする場合は、SWF ファイルをホストするサーバーに " クロスドメインポリシー ファイル " を置いておく必要があります。

sendAndLoad() メソッドを実行すると、XML オブジェクトの loaded プロパティが false に設定されます。XML データのダウンロードが終了すると、データが正常にロードされた場合は loaded プロパティが true に設定され、onLoad イベントハンドラが呼び出されます。XML データは、完全にダウンロードされるまで解析されません。XML オブジェクトに XML ツリーが既に含まれていた場合、その XML ツリーは破棄されます。

パラメータ

`url:String` - ストリング。指定された XML オブジェクトの宛先 URL です。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、`url` は SWF ファイルと同じドメインに属している必要があります。詳細については、「説明」を参照してください。

`resultXML:XML` - XML コンストラクタメソッドで作成されたターゲット XML オブジェクトで、このオブジェクトはサーバーから返される情報を受信します。

例

次の例には、簡単な電子商取引店頭アプリケーション用の ActionScript が含まれています。

`XML.sendAndLoad()` メソッドは、ユーザーの名前とパスワードを含む XML エレメントを転送し、`onLoad` ハンドラを使用してサーバーからの応答を処理します。

```
var login_str:String = "<login username=\""+username_txt.text+"\" password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = myOnLoad;
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",
    myLoginReply_xml);
function myOnLoad(success:Boolean) {
    if (success) {
        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }
    } else {
        gotoAndStop("connectionFailed");
    }
}
```

関連項目

`send` (`XML.send` メソッド), `load` (`XML.load` メソッド), `loaded` (`XML.loaded` プロパティ), `onLoad` (`XML.onLoad` ハンドラ)

status (XML.status プロパティ)

```
public status : Number
```

XML ドキュメントが XML オブジェクトに正常に解析されたかどうかを示す数値を自動的に設定し、返します。数値ステータスコードとその説明を次に示します。

- 0 エラーなし。解析が正常に終了しました。
- -2 CDATA セクションが適切に終了されていません。
- -3 XML 宣言が適切に終了されていません。
- -4 DOCTYPE 宣言が適切に終了されていません。
- -5 コメントが適切に終了されていません。
- -6 XML エレメントの形式が正しくありませんでした。
- -7 メモリ不足です。
- -8 属性値が適切に終了されていません。
- -9 開始タグに対応する終了タグがありません。
- -10 対応する開始タグのない終了タグが見つかりました。

例

次の例では、SWF ファイルに XML パケットをロードします。XML のロードと解析が成功したかどうかに応じて、ステータスマッセージが表示されます。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_xml:XML = new XML();
my_xml.onLoad = function(success:Boolean) {
    if (success) {
        if (my_xml.status == 0) {
            trace("XML was loaded and parsed successfully");
        } else {
            trace("XML was loaded successfully, but was unable to be parsed.");
        }
        var errorMessage:String;
        switch (my_xml.status) {
        case 0 :
            errorMessage = "No error; parse was completed successfully.";
            break;
        case -2 :
            errorMessage = "A CDATA section was not properly terminated.";
            break;
        case -3 :
            errorMessage = "The XML declaration was not properly terminated.";
            break;
        case -4 :
            errorMessage = "The DOCTYPE declaration was not properly terminated.";
            break;
        }
    }
}
```

```

        case -5 :
            errorMessage = "A comment was not properly terminated.";
            break;
        case -6 :
            errorMessage = "An XML element was malformed.";
            break;
        case -7 :
            errorMessage = "Out of memory.";
            break;
        case -8 :
            errorMessage = "An attribute value was not properly terminated.";
            break;
        case -9 :
            errorMessage = "A start-tag was not matched with an end-tag.";
            break;
        case -10 :
            errorMessage = "An end-tag was encountered without a matching
            start-tag.";
            break;
        default :
            errorMessage = "An unknown error has occurred.";
            break;
    }
    trace("status: "+my_xml.status+" ("+errorMessage+ ")");
} else {
    trace("Unable to load/parse XML. (status: "+my_xml.status+ ")");
}
};

my_xml.load("http://www.helpexamples.com/flash/badxml.xml");

```

XML コンストラクタ

`public XML(text:String)`

新しい XML オブジェクトを作成します。XML クラスのメソッドを呼び出す前に、コンストラクタを使用して XML オブジェクトを作成する必要があります。

メモ: `createElement()` メソッドおよび `createTextNode()` メソッドを使用して、XML ドキュメントツリーにエレメントとテキストノードを追加します。

パラメータ

`text:String` - ストリング。新しい XML オブジェクトを作成するために解析される XML テキスト。

例

次の例では、新しい空の XML オブジェクトを作成します。

```
var my_xml:XML = new XML();
```

次の例では、source パラメータで指定された XML テキストを解析することによって XML オブジェクトを作成し、新しく作成した XML オブジェクトに XML ドキュメントツリーを設定します。

```
var other_xml:XML = new XML("<state name=\"California\"><city>San Francisco</city></state>");
```

関連項目

[createElement \(XML.createElement メソッド\)](#),
[createTextNode \(XML.createTextNode メソッド\)](#)

xmlDecl (XML.xmlDecl プロパティ)

```
public xmlDecl : String
```

ドキュメントの XML 宣言についての情報を指定するストリング。XML ドキュメントが XML オブジェクトに解析された後、このプロパティはドキュメントの XML 宣言のテキストに設定されます。このプロパティは、XML ノードオブジェクトでなく XML 宣言のストリング表現を使用して設定されます。解析中に XML 宣言が見つからなかった場合、プロパティは undefined.XML に設定されます。

XML.toString() メソッドは、XML オブジェクト内の他のテキストの前に、XML.xmlDecl プロパティの内容を出力します。XML.xmlDecl プロパティに undefined タイプが含まれている場合、XML 宣言は出力されません。

例

次の例では、ステージと同じサイズのテキストフィールド my_txt を作成します。このテキストフィールドには、SWF ファイルにロードされる XML パケットのプロパティが表示されます。ドキュメントタイプ宣言が my_txt に表示されます。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "_typewriter";
my_fmt.size = 12;
my_fmt.leftMargin = 10;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, Stage.width,
    Stage.height);
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.setNewTextFormat(my_fmt);

var my_xml:XML = new XML();
```

```

my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    var endTime:Number = getTimer();
    var elapsedTime:Number = endTime-startTime;
    if (success) {
        my_txt.text = "xmlDecl:"+newline+my_xml.xmlDecl+newline+newline;
        my_txt.text += "contentType:"+newline+my_xml.contentType+newline+newline;
        my_txt.text += "docTypeDecl:"+newline+my_xml.docTypeDecl+newline+newline;
        my_txt.text += "packet:"+newline+my_xml.toString()+newline+newline;
    } else {
        my_txt.text = "Unable to load remote XML."+newline+newline;
    }
    my_txt.text += "loaded in: "+elapsedTime+" ms.";
};

my_xml.load("http://www.helpexamples.com/crossdomain.xml");
var startTime:Number = getTimer();

```

関連項目

[docTypeDecl \(XML.docTypeDecl プロパティ\)](#)

XMLNode

[Object](#)

```

|  
+- XMLNode
```

```

public class XMLNode  
extends Object
```

XML ドキュメントは、Flash では XML クラスで表現されます。階層ドキュメントの各エレメントが、[XMLNode](#) オブジェクトで表されます。

関連項目

[XML](#)

プロパティ一覧

オプション	プロパティ	説明
	attributes:Object	指定された XML インスタンスのすべての属性を含むオブジェクト。
	childNodes:Array (読み取り専用)	指定された XML オブジェクトの子の配列。
	firstChild:XMLNode (読み取り専用)	指定された XML オブジェクトを評価し、親ノードの子リスト内の最初の子を参照します。

オプション	プロパティ	説明
	<code>lastChild:XMLNode</code> (読み取り専用)	ノードの子リスト内の最後の子を参照する XMLNode 値。
	<code>nextSibling:XMLNode</code> (読み取り専用)	親ノードの子リスト内の次の子ノードを参照する XMLNode 値。
	<code>nodeName:String</code>	XML オブジェクトのノード名を表すストリング。
	<code>nodeType:Number</code> (読み取り専用)	nodeType の値。 XML エレメントの場合は 1、テキストノードの場合は 3 になります。
	<code>nodeValue:String</code>	XML オブジェクトのノード値。
	<code>parentNode:XMLNode</code> (読み取り専用)	指定された XML オブジェクトの親ノードを参照する XMLNode 値。ノードに親がない場合は null を返します。
	<code>previousSibling:</code> <code>XMLNode</code> (読み取り専用)	親ノードの子リスト内の前の子ノードを参照する XMLNode 値。

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

メソッド一覧

オプション	シグネチャ	説明
	<code>appendChild(newChild: XMLNode) : Void</code>	指定されたノードを XML オブジェクトの子リストに追加します。
	<code>cloneNode (deep:Boolean) : XMLNode</code>	指定された XML オブジェクトと同じタイプ、名前、値、および属性を持つ新しい XML ノードを作成し、返します。
	<code>hasChildNodes() : Boolean</code>	XML オブジェクトに子ノードがあるかどうかを指定します。
	<code>insertBefore(newChild: XMLNode, insertPoint: XMLNode) : Void</code>	insertPoint ノードの前に newChild ノードを XML オブジェクトの子リストに挿入します。
	<code>removeNode() : Void</code>	指定された XML オブジェクトをその親から削除します。
	<code>toString() : String</code>	指定された XML オブジェクトを評価し、ノード、子、および属性を含む XML 構造体のテキスト表現を作成し、結果をストリングとして返します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

appendChild (XMLNode.appendChild メソッド)

public appendChild(newChild:[XMLNode](#)) : Void

指定されたノードを XML オブジェクトの子リストに追加します。このメソッドは、パラメータ childNode で参照されるノードに直接実行されます。ノードのコピーが追加されるわけではありません。追加するノードが別のツリー構造内に既に存在している場合は、ノードを新しい場所に追加すると現在の場所からノードが削除されます。パラメータ childNode が、別の XML ツリー構造内に既に存在しているノードを参照している場合、追加される子ノードは、今までの親ノードから削除された後で新しいツリー構造に置かれます。

パラメータ

newChild:[XMLNode](#) - 現在の場所から my_xml オブジェクトの子リストに移動されるノードを表す XMLNode。

例

この例では、次の処理を記載されている順序で実行します。

- 2つの空の XML ドキュメント、doc1 および doc2 を作成します。
- createElement() メソッドを使用して新しいノードを作成し、appendChild() メソッドを使用してそのノードを XML ドキュメント doc1 に追加します。
- ルートノードを doc1 から doc2 に移動して、appendChild() メソッドを使用してノードを移動する方法を示します。
- doc2 からルートノードを複製し、そのノードを doc1 に追加します。
- 新しいノードを作成し、それを XML ドキュメント doc1 のルートノードに追加します。

```
var doc1:XML = new XML();  
var doc2:XML = new XML();  
  
// create a root node and add it to doc1  
var rootnode:XMLNode = doc1.createElement("root");  
doc1.appendChild(rootnode);  
trace ("doc1: " + doc1); // output: doc1: <root />  
trace ("doc2: " + doc2); // output: doc2:
```

```

// move the root node to doc2
doc2.appendChild(roottnode);
trace ("doc1: " + doc1); // output: doc1:
trace ("doc2: " + doc2); // output: doc2: <root />

// clone the root node and append it to doc1
var clone:XMLNode = doc2.firstChild.cloneNode(true);
doc1.appendChild(clone);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2: <root />

// create a new node to append to root node (named clone) of doc1
var newNode:XMLNode = doc1.createElement("newbie");
clone.appendChild(newNode);
trace ("doc1: " + doc1); // output: doc1: <root><newbie /></root>

```

attributes (XMLNode.attributes プロパティ)

public attributes : Object

指定された XML インスタンスのすべての属性を含むオブジェクト。XML.attributes オブジェクトには、XML インスタンスの各属性に対して1つずつ変数があります。この変数はオブジェクトの一部として定義されているので、通常はオブジェクトのプロパティとして参照されます。各属性の値は、対応するプロパティにストリングとして保存されます。たとえば、color という名前の属性がある場合、次のコードで示すように、color をプロパティ名として指定して、その属性の値を取得することができます。

```
var myColor:String = doc.firstChild.attributes.color
```

例

次の例では、XML 属性名を表示します。

```

// create a tag called 'mytag' with
// an attribute called 'name' with value 'Val'
var doc:XML = new XML("<mytag name=\"Val\"> item </mytag>");

// assign the value of the 'name' attribute to variable y
var y:String = doc.firstChild.attributes.name;
trace (y); // output: Val

// create a new attribute named 'order' with value 'first'
doc.firstChild.attributes.order = "first";

// assign the value of the 'order' attribute to variable z
var z:String = doc.firstChild.attributes.order
trace(z); // output: first

```

次の情報が [出力] パネルに表示されます。

```
Val  
first
```

childNodes (XMLNode.childNodes プロパティ)

```
public childNodes : Array (読み取り専用)
```

指定された XML オブジェクトの子の配列。配列内の各エレメントは、子ノードを表す XML オブジェクトへの参照です。これは読み取り専用プロパティであり、子ノードを操作する場合には使用できません。子ノードを操作するには、appendChild()、insertBefore()、および removeNode() メソッドを使用します。

テキストノード (nodeType == 3) の場合、このプロパティは undefined になります。

例

次の例では、XML.childNodes プロパティを使用して子ノードの配列を返す方法を示します。

```
// create a new XML document  
var doc:XML = new XML();  
  
// create a root node  
var rootNode:XMLNode = doc.createElement("rootNode");  
  
// create three child nodes  
var oldest:XMLNode = doc.createElement("oldest");  
var middle:XMLNode = doc.createElement("middle");  
var youngest:XMLNode = doc.createElement("youngest");  
  
// add the rootNode as the root of the XML document tree  
doc.appendChild(rootNode);  
  
// add each of the child nodes as children of rootNode  
rootNode.appendChild(oldest);  
rootNode.appendChild(middle);  
rootNode.appendChild(youngest);  
  
// create an array and use rootNode to populate it  
var firstArray:Array = doc.childNodes;  
trace(firstArray);  
// output: <rootNode><oldest /><middle /><youngest /></rootNode>  
  
// create another array and use the child nodes to populate it  
var secondArray:Array = rootNode.childNodes;  
trace(secondArray);  
// output: <oldest />,<middle />,<youngest />
```

関連項目

[nodeType](#) (`XMLNode.nodeType` プロパティ), [appendChild](#) (`XMLNode.appendChild` メソッド), [insertBefore](#) (`XMLNode.insertBefore` メソッド), [removeNode](#) (`XMLNode.removeNode` メソッド)

cloneNode (`XMLNode.cloneNode` メソッド)

`public cloneNode(deep:Boolean) : XMLNode`

指定された XML オブジェクトと同じタイプ、名前、値、および属性を持つ新しい XML ノードを作成し、返します。deep に true を設定すると、すべての子ノードのクローンが再帰的に作成されるため、元のオブジェクトのドキュメントツリーが正確に複製されます。

返されたノードのクローンは、クローン作成元のアイテムのツリーとは関連がなくなります。その結果、`nextSibling`、`parentNode`、および `previousSibling` の値はすべて null になります。deep パラメータが false の場合、または `my_xml` ノードに子ノードがない場合は、`firstChild` と `lastChild` も null です。

パラメータ

`deep:Boolean` - ブール値。true に設定した場合、指定された XML オブジェクトの子のクローンが再帰的に作成されます。

戻り値

`XMLNode` - `XMLNode` オブジェクト。

例

次の例では、`XML.cloneNode()` メソッドを使用してノードのコピーを作成する方法を示します。

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
```

```

rootNode.appendChild(youngest);

// create a copy of the middle node using cloneNode()
var middle2:XMLNode = middle.cloneNode(false);

// insert the clone node into rootNode between the middle and youngest nodes
rootNode.insertBefore(middle2, youngest);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>

// create a copy of rootNode using cloneNode() to demonstrate a deep copy
var rootClone:XMLNode = rootNode.cloneNode(true);

// insert the clone, which contains all child nodes, to rootNode
rootNode.appendChild(rootClone);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>
// </rootNode>

```

firstChild (XMLNode.firstChild プロパティ)

public firstChild : XMLNode (読み取り専用)

指定された XML オブジェクトを評価し、親ノードの子リスト内の最初の子を参照します。このプロパティは、ノードに子がないときは null です。ノードがテキストノードの場合、このプロパティは undefined です。これは読み取り専用プロパティであり、子ノードを操作する場合には使用できません。子ノードを操作するには、appendChild() メソッド insertBefore() メソッド、および removeNode() メソッドを使用します。

例

次の例では、XML.firstChild を使用してノードの子ノードをループ処理する方法を示します。

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use firstChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
    aNode.nextSibling) {
    trace(aNode);
}

// output:
// <oldest />
// <middle />
// <youngest />
```

次の例は、"Examples" ディレクトリの "XML_languagePicker" FLA ファイルから採られています。これは languageXML.onLoad イベントハンドラ関数定義にあります。

```
// loop through the strings in each language node
// adding each string as a new element in the language array
for (var stringNode:XMLNode = childNode.firstChild; stringNode != null;
    stringNode = stringNode.nextSibling, j++) {
    masterArray[i][j] = stringNode.firstChild.nodeValue;
}
```

スクリプト全体を表示するには、www.adobe.com/go/learn_fl_samples_jp の "ActionScript" サンプルフォルダにある "XML_languagePicker.fla" ファイルを参照してください。.zip ファイルをダウンロードして解凍し、ActionScript バージョンのフォルダに移動してサンプルにアクセスします。

関連項目

[appendChild \(XMLNode.appendChild メソッド\)](#), [insertBefore \(XMLNode.insertBefore メソッド\)](#), [removeNode \(XMLNode.removeNode メソッド\)](#)

hasChildNodes (XMLNode.hasChildNodes メソッド)

public hasChildNodes() : Boolean

XML オブジェクトに子ノードがあるかどうかを指定します。

戻り値

Boolean - 指定された XML オブジェクトに子ノードがある場合は true、それ以外の場合は false。

例

次の例では、新しい XML パケットを作成します。このコードは、ルートノードに子ノードがある場合は各子ノードをループして、そのノードの名前と値を表示します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
    // use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
        aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

次の情報が [出力] パネルに表示されます。

```
output:
username: hank
password: rudolph
```

insertBefore (XMLNode.insertBefore メソッド)

public insertBefore(newChild:XMLNode, insertPoint:XMLNode) : Void

insertPoint ノードの前に newChild ノードを XML オブジェクトの子リストに挿入します。
insertPoint が XMLNode オブジェクトの子でない場合、挿入は失敗します。

パラメータ

newChild:XMLNode - 挿入される XMLNode オブジェクト。

insertPoint:XMLNode - メソッドの呼び出し後に newChild ノードに従う XMLNode オブジェクト。

例

次の例では、2 つの既存のノード間に新しい XML ノードを挿入します。

```
var my_xml:XML = new XML("<a>1</a>\n<c>3</c>");
var insertPoint:XMLNode = my_xml.lastChild;
```

```
var newNode:XML = new XML("<b>2</b>\n");
my_xml.insertBefore(newNode, insertPoint);
trace(my_xml);
```

関連項目

[XML.cloneNode \(XMLNode.cloneNode メソッド\)](#)

lastChild (XMLNode.lastChild プロパティ)

public lastChild : [XMLNode](#) (読み取り専用)

ノードの子リスト内の最後の子を参照する [XMLNode](#) 値。[XML.lastChild](#) プロパティは、ノードに子がないときは `null` です。このプロパティは子ノードの操作には使用できません。子ノードを操作するには、[appendChild\(\)](#) メソッド、[insertBefore\(\)](#) メソッド、および [removeNode\(\)](#) メソッドを使用します。

例

次の例では、[XML.lastChild](#) プロパティを使用して XML ノードの子ノードで繰り返し処理を実行します。この処理は、ノードの子リストの最後の項目から始まり、ノードの子リストの最初の子で終了します。

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
    aNode.previousSibling) {
    trace(aNode);
}

// output:
```

```

// <youngest />
// <middle />
// <oldest />

次の例では、新しい XML パケットを作成し、XML.lastChild プロパティを使用して、ルートノードの子ノードで繰り返し処理を実行します。

// create a new XML document
var doc:XML = new XML("");

var rootNode:XMLNode = doc.firstChild;

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null;
    aNode=aNode.previousSibling) {
    trace(aNode);
}

// output:
// <youngest />
// <middle />
// <oldest />

```

関連項目

[appendChild \(XMLNode.appendChild メソッド\)](#), [insertBefore \(XMLNode.insertBefore メソッド\)](#), [removeNode \(XMLNode.removeNode メソッド\)](#), [XML](#)

nextSibling (XMLNode.nextSibling プロパティ)

public nextSibling : [XMLNode](#) (読み取り専用)

親ノードの子リスト内の次の子ノードを参照する XMLNode 値。子リストに次のノードがない場合、このプロパティは null になります。このプロパティは子ノードの操作には使用できません。子ノードを操作するには、`appendChild()` メソッド、`insertBefore()` メソッド、および `removeNode()` メソッドを使用します。

例

次に示す例は、`XML.firstChild` プロパティの例から引用したもので、`XML.nextSibling` プロパティを使用して XML ノードの子ノードをループ処理する方法を示しています。

```

for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
    aNode.nextSibling) {
    trace(aNode);
}

```

関連項目

[firstChild \(XMLNode.firstChild プロパティ\)](#), [appendChild \(XMLNode.appendChild メソッド\)](#), [insertBefore \(XMLNode.insertBefore メソッド\)](#), [removeNode \(XMLNode.removeNode メソッド\)](#), [XML](#)

nodeName (XMLNode.nodeName プロパティ)

public nodeName : String

XML オブジェクトのノード名を表すストリング。XML オブジェクトが XML エレメントである場合 (nodeType == 1)、nodeName は XML ファイル内のノードを表すタグの名前です。たとえば、TITLE は HTML TITLE タグの nodeName です。XML オブジェクトがテキストノードである (nodeType == 3) 場合、nodeName は null です。

例

次の例では、エレメントノードとテキストノードを作成し、それぞれのノード名をチェックします。

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeName);
trace(myTextNode.nodeName);

// output:
// rootNode
// null
```

次の例では、新しい XML パケットを作成します。このコードは、ルートノードに子ノードがある場合は各子ノードをループして、そのノードの名前と値を表示します。次の ActionScript を FLA ファイルまたは AS ファイルに追加します。

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
    // use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
        aNode=aNode.nextSibling) {
```

```

        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}

```

次のノード名が [出力] パネルに表示されます。

```

output:
username: hank
password: rudolph

```

関連項目

[nodeType \(XMLNode.nodeType プロパティ \)](#)

nodeType (XMLNode.nodeType プロパティ)

public `nodeType : Number` (読み取り専用)

`nodeType` の値。XML エレメントの場合は 1、テキストノードの場合は 3 になります。

`nodeType` は、W3C DOM レベル 1 勧告 (www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html) の `NodeType` の一覧に記載されている数値です。次の表は、これらの数値の一覧です。

整数値	定義されている定数
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

Flash Player に組み込まれている XML クラスがサポートするのは 1(ELEMENT_NODE) と 3 (TEXT_NODE) だけです。

例

次の例では、エレメントノードとテキストノードを作成し、それぞれのノードタイプをチェックします。

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeType);
trace(myTextNode.nodeType);

// output:
// 1
// 3
```

関連項目

[nodeValue \(XMLNode.nodeValue プロパティ \)](#)

nodeValue (XMLNode.nodeValue プロパティ)

public `nodeValue : String`

XML オブジェクトのノード値。XML オブジェクトがテキストノードである場合、`nodeType` は 3 であり、`nodeValue` はノードのテキストです。XML オブジェクトが XML エレメントである場合 (`nodeType` が 1)、`nodeValue` は `null` で、読み取り専用です。

例

次の例では、エレメントノードとテキストノードを作成し、それぞれのノード値をチェックします。

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
```

```
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeValue);
trace(myTextNode.nodeValue);

// output:
// null
// myTextNode
```

次の例では、XML パケットを作成して解析します。このコードは各子ノードをループ処理し、`firstChild` プロパティと `firstChild.nodeValue` を使用してノード値を表示します。`firstChild` を使用してノードの内容を表示した場合は、& エンティティが保持されます。しかし、明示的に `nodeValue` を使用した場合は、アンパサンド文字 (&) に変換されます。

```
var my_xml:XML = new XML("mortongood&evil");
trace("using firstChild:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild);
}
trace("");
trace("using firstChild.nodeValue:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild.nodeValue);
}
```

次の情報が [出力] パネルに表示されます。

```
using firstChild:
    morton
    good&evil

using firstChild.nodeValue:
    morton
    good&evil
```

関連項目

[nodeType \(XMLNode.nodeType プロパティ\)](#)

parentNode (XMLNode.parentNode プロパティ)

public parentNode : XMLNode (読み取り専用)

指定された XML オブジェクトの親ノードを参照する XMLNode 値。ノードに親がない場合は null を返します。これは読み取り専用プロパティであり、子ノードを操作する場合には使用できません。子ノードを操作するには、appendChild() メソッド insertBefore() メソッド、および removeNode() メソッドを使用します。

例

次の例では、XML パケットを作成し、username ノードの親ノードを [出力] パネルに表示します。

```
var my_xml:XML = new XML("morton&good&evil");

// first child is the <login /> node
var rootNode:XMLNode = my_xml.firstChild;

// first child of the root is the <username /> node
var targetNode:XMLNode = rootNode.firstChild;
trace("the parent node of '" + targetNode.nodeName + "' is:
      "+targetNode.parentNode.nodeName);
trace("contents of the parent node are:\n"+targetNode.parentNode);

// output (line breaks added for clarity):

the parent node of 'username' is: login
contents of the parent node are:
  morton
  good&evil
```

関連項目

[appendChild \(XMLNode.appendChild メソッド\), insertBefore \(XMLNode.insertBefore メソッド\), removeNode \(XMLNode.removeNode メソッド\), XML](#)

previousSibling (XMLNode.previousSibling プロパティ)

public previousSibling : XMLNode (読み取り専用)

親ノードの子リスト内の前の子ノードを参照する XMLNode 値。ノードに前の兄弟ノードがない場合は、null を返します。このプロパティは子ノードの操作には使用できません。子ノードを操作するには、appendChild() メソッド、insertBefore() メソッド、および removeNode() メソッドを使用します。

例

次に示す例は、`XML.lastChild` プロパティの例から引用したもので、`XML.previousSibling` プロパティを使用して XML ノードの子ノードをループ処理する方法を示しています。

```
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
    aNode.previousSibling) {
    trace(aNode);
}
```

関連項目

`lastChild` (`XMLNode.lastChild` プロパティ), `appendChild` (`XMLNode.appendChild` メソッド), `insertBefore` (`XMLNode.insertBefore` メソッド), `removeNode` (`XMLNode.removeNode` メソッド), `XML`

removeNode (`XMLNode.removeNode` メソッド)

`public removeNode():Void`

指定された XML オブジェクトをその親から削除します。また、ノードのすべての子孫も削除します。

例

次の例では、XML パケットを作成し、指定された XML オブジェクトとその子孫ノードを削除します。

```
var xml_str:String = "<state name=\"California\"><city>San Francisco</city></
state>";

var my_xml:XML = new XML(xml_str);
var cityNode:XMLNode = my_xml.firstChild.firstChild;
trace("before XML.removeNode():\n"+my_xml);
cityNode.removeNode();
trace("");
trace("after XML.removeNode():\n"+my_xml);

// output (line breaks added for clarity):
//
// before XML.removeNode():
// <state name="California">
// <city>San Francisco</city>
// </state>
//
// after XML.removeNode():
// <state name="California" />
```

toString (XMLNode.toString メソッド)

```
public toString() : String
```

指定された XML オブジェクトを評価し、ノード、子、および属性を含む XML 構造体のテキスト表現を作成し、結果をストリングとして返します。

トップレベルの XML オブジェクト (コンストラクタで作成されたオブジェクト) の場合は、
XML.toString() メソッドがドキュメントの XML 宣言 (保存場所は XML.xmlDecl プロパティ)
を出力し、その後にドキュメントの DOCTYPE 宣言 (保存場所は XML.docTypeDecl プロパティ) を
続け、さらにオブジェクト内のすべての XML ノードのテキスト表現を続けます。 XML 宣言は、
XML.xmlDecl プロパティが undefined の場合は出力されません。 DOCTYPE 宣言は、
XML.docTypeDecl プロパティが undefined の場合は出力されません。

戻り値

[String](#) - ストリング。

例

次のコードでは、`toString()` メソッドを使用して `XMLNode` オブジェクトをストリングに変換し、
次に `String` クラスの `toUpperCase()` メソッドを使用します。

```
var xString = "<first>Mary</first>"  
+ "<last>Ng</last>"  
var my_xml:XML = new XML(xString);  
var my_node:XMLNode = my_xml.childNodes[1];  
trace(my_node.toString().toUpperCase());  
// <LAST>NG<
```

関連項目

[docTypeDecl](#) (`XML.docTypeDecl` プロパティ), [xmlDecl](#) (`XML.xmlDecl` プロパティ)

XMLSocket

```
Object
|
+-XMLSocket
```

```
public class XMLSocket
extends Object
```

XMLSocket クラスはクライアントソケットを実装しており、Flash Lite プレーヤーを実行するデバイスはこのソケットを使用して、IP アドレスまたはドメイン名で識別されるサーバーコンピュータと通信することができます。XMLSocket クラスは、リアルタイムのチャットシステムなど待ち時間を短くすることが求められるクライアント / サーバーアプリケーションに適しています。従来の HTTP ベースチャットシステムは頻繁にサーバーをポーリングし、HTTP 要求を使用して新しいメッセージをダウンロードします。それに対して、XMLSocket チャットソリューションはサーバーに対して開いた接続を維持するため、サーバーはクライアントから要求を受けずに直ちに着信メッセージを送ることができます。

XMLSocket クラスを使用するには、サーバーコンピュータは XMLSocket クラスで使用されるプロトコルに対応したデーモンプロセスを実行する必要があります。プロトコルの説明を次の一覧に示します。

- XML メッセージは、全二重 TCP/IP ストリームソケット接続を介して送られます。
- 各 XML メッセージは、ゼロ (0) バイトで終了する完全な XML ドキュメントです。
- 1つの XMLSocket 接続を使用して送受信できる XML メッセージの数に制限はありません。

XMLSocket オブジェクトがサーバーに接続する方法と場所については、次の制限があります。

- XMLSocket を 1024 以下のポートに接続するには、アプリケーションが独自の正確なドメインに接続している場合でも、最初に `System.security.loadPolicyFile()` メソッドを使用してポリシーファイルをロードする必要があります。
- `XMLSocket.connect()` メソッドは、SWF ファイルが存在するのと同じドメイン内のコンピュータにしか接続できません。この制限は、ローカルで再生される SWF ファイルには適用されません。この制限は、`loadVariables()` 関数、`XML.sendAndLoad()` メソッド、および `XML.load()` メソッドのセキュリティ規則と同じです。特定のドメインからのアクセスを許可するセキュリティポリシーファイルを作成すると、SWF ファイルが存在するドメイン以外で実行されるサーバーデーモンに接続できます。

XMLSocket オブジェクトと通信するようにサーバーを設定すると、問題が発生する可能性があります。アプリケーションがリアルタイムのインタラクティブ機能を必要としない場合は、XMLSocket クラスの代わりに、`loadVariables()` 関数、または Flash の HTTP ベース XML サーバー接続メソッド (`XML.load()`、`XML.sendAndLoad()`、`XML.send()`) を使用します。

XMLSocket クラスのメソッドを使用するには、まず `XMLSocket()` コンストラクタを使用して XMLSocket オブジェクトを作成する必要があります。

関連項目

[loadPolicyFile \(security.loadPolicyFile メソッド\)](#)

プロパティ一覧

Object クラスから継承されるプロパティ

```
constructor (Object.constructor プロパティ), __proto__ (Object.__proto__ プロ  
パティ), prototype (Object.prototype プロパティ), __resolve (Object.__resolve  
プロパティ)
```

イベント一覧

イベント	説明
<code>onClose = function() {}</code>	開いた接続がサーバーによって閉じられたときにだけ呼び出されます。
<code>onConnect = function(success: Boolean) {}</code>	XMLSocket.connect() の成功または失敗により接続要求が開始されたときに、Flash Lite Player が呼び出す非同期コールバックです。
<code>onData = function (src:String) {}</code>	ゼロ (0) バイトで終了する XML メッセージがサーバーからダウンロードされると、呼び出されます。
<code>onXML = function(src:XML) {} {} {}</code>	XML ドキュメントを含む指定の XML オブジェクトが、開いている XMLSocket 接続を通して届いたときに、Flash Lite Player によって呼び出されます。

コンストラクター一覧

シグネチャ	説明
<code>XMLSocket()</code>	新しい XMLSocket オブジェクトを作成します。

メソッド一覧

オプション	シグネチャ	説明
	<code>close() : Void</code>	XMLSocket オブジェクトで指定された接続を閉じます。
	<code>connect(url:String, port:Number) : Boolean</code>	指定された TCP ポートを使用して指定されたインターネットホストへの接続を確立し、接続が正常に開始されたかどうかにより true または false を返します。
	<code>send(data:Object) : Void</code>	object パラメータで指定された XML オブジェクトまたはデータをストリングに変換し、その後ろにゼロ (0) バイトを付加してサーバーに転送します。

Object クラスから継承されるメソッド

```
addProperty (Object.addProperty メソッド), hasOwnProperty  
(Object.hasOwnProperty メソッド), isPrototypeOf  
(Object.isPrototypeOf メソッド), isPrototypeOfOf (Object.isPrototypeOfOf  
メソッド), registerClass (Object.registerClass メソッド), toString  
(Object.toString メソッド), unwatch (Object.unwatch メソッド), valueOf  
(Object.valueOf メソッド), watch (Object.watch メソッド)
```

close (XMLSocket.close メソッド)

public close() : Void

XMLSocket オブジェクトで指定された接続を閉じます。

例

次の簡単な例では、XMLSocket オブジェクトを作成し、サーバーへの接続を行ってから、接続を閉じます。

```
var socket:XMLSocket = new XMLSocket();  
socket.connect(null, 2000);  
socket.close();
```

関連項目

[connect \(XMLSocket.connect メソッド \)](#)

connect (XMLSocket.connect メソッド)

public connect(url:String, port:Number) : Boolean

指定された TCP ポートを使用して指定されたインターネットホストへの接続を確立し、接続が正常に開始されたかどうかにより true または false を返します。XMLSocket.connect() メソッドが返す値が true である場合、接続プロセスの初期段階は成功です。この後で、最終的な接続が成功したか失敗したかを判断するために XMLSocket.onConnect() メソッドが呼び出されます。

XMLSocket.connect() メソッドが false を返した場合は、接続に失敗しています。

インターネットホストコンピュータのポート番号がわからない場合は、ネットワーク管理者に問い合わせてください。XMLSocket を 1024 以下のポートに接続するには、最初に

System.security.loadPolicyFile() メソッドを使用してポリシーファイルをロードする必要があります。

host パラメータに対して null を指定した場合は、XMLSocket.connect() を呼び出す SWF ファイルが存在するホストに接続します。たとえば、www.example.com から SWF ファイルをダウンロードした場合、host パラメータに null を指定することは www.example.com の IP アドレスを入力することと同じです。

Flash Player 7 以降で SWF ファイルを実行している場合、host パラメータは正確に同じドメインに置かれている必要があります。たとえば、www.someDomain.com に置かれている SWF ファイルが、Flash Player 5 用にパブリッシュされていて、Flash Player 7 以降で実行される場合、www.someDomain.com に存在する SWF ファイルからしか変数をロードできません。異なるドメインから変数をロードする場合は、アクセスされる側の SWF ファイルをホスティングするサーバーにクロスドメインポリシーファイルを置いておく必要があります。

メモ: System.capabilities.hasXMLSocket が false の場合は、XMLSocket.connect() メソッドは false を返します。

パラメータ

url:String - ストリング。DNS の完全修飾ドメイン名、つまり aaa.bbb.ccc.ddd という形式の IP アドレスです。SWF ファイルが存在するホストサーバーに接続するために、null を指定することもできます。呼び出し元の SWF ファイルが Web ブラウザで実行されている場合、host は SWF ファイルと同じドメインに属している必要があります。

port:Number - 数値。接続の確立に使用するホスト上の TCP ポート番号です。

戻り値

Boolean - 接続に成功した場合は true を返します。それ以外の場合は false を返します。

例

次の例では、SWF ファイルが存在するホストに XMLSocket.connect() メソッドを使用して接続し、接続が成功したか失敗したかを示す戻り値を trace() 関数を使用して表示します。

```
var socket:XMLSocket = new XMLSocket();
socket.onConnect = function (success:Boolean) {
    if (success) {
        trace ("Connection succeeded!");
    } else {
        trace ("Connection failed!");
    }
}
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!");
}
```

関連項目

[onConnect \(XMLSocket.onConnect ハンドラ\)](#), [function ステートメント](#), [loadPolicyFile \(security.loadPolicyFile メソッド\)](#)

onClose (XMLSocket.onClose ハンドラ)

```
onClose = function() {}
```

開いた接続がサーバーによって閉じられたときにだけ呼び出されます。このメソッドのデフォルトの実行形態では、アクションを実行しません。デフォルトの実行形態を無効にするには、カスタムアクションを含む関数を割り当てます。

例

次の例では、開いた接続がサーバーによって閉じられた場合に、trace ステートメントを実行します。

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.onClose = function () {
    trace("Connection to server lost.");
}
```

関連項目

[onConnect \(XMLSocket.onConnect ハンドラ \),function ステートメント](#)

onConnect (XMLSocket.onConnect ハンドラ)

```
onConnect = function(success:Boolean) {}
```

XMLSocket.connect() の成功または失敗により接続要求が開始されたときに、Flash Lite Player が呼び出す非同期コールバックです。接続が成功した場合、success パラメータの値は true に設定され、失敗した場合は false に設定されます。

このメソッドのデフォルトの実行形態では、アクションを実行しません。デフォルトの実行形態を無効にするには、カスタムアクションを含む関数を割り当てます。

パラメータ

success:Boolean - ソケット接続が正常に確立されているかどうかを示すブール値。接続が成功した場合、success パラメータの値は true に設定され、失敗した場合は false に設定されます。

例

次の例では、簡単なチャットアプリケーションで onConnect() メソッドに対して置換関数を指定するプロセスを示します。

このスクリプトは、コンストラクタメソッドで XMLSocket オブジェクトを作成した後、`onConnect()` イベントハンドラが呼び出されたときに実行されるカスタム関数を定義します。この関数は、接続が正常に確立されたかどうかに応じて、どの画面を表示するかを制御します。接続が正常に確立されると、`startChat` というフレーム上のメインのチャット画面が表示されます。接続が失敗すると、`connectionFailed` というフレーム上のトラブルシューティング情報を示す画面が表示されます。

```
var socket:XMLSocket = new XMLSocket();
socket.onConnect = function (success) {
    if (success) {
        gotoAndPlay("startChat");
    } else {
        gotoAndStop("connectionFailed");
    }
}
```

`onConnect()` ハンドラを定義すると、`connect()` メソッドが呼び出されて、接続が確立されます。`connect()` メソッドが `false` を返すと、SWF ファイルが `connectionFailed` というフレームに直接送られ、`onConnect()` は呼び出されません。`connect()` メソッドが `true` を返すと、SWF ファイルは "Please wait" 画面の `waitForConnection` というフレームにジャンプします。SWF ファイルは、`onConnect()` ハンドラが呼び出されるまで `waitForConnection` フレーム上にあります。この呼び出しがいつ行われるかは、ネットワークの待ち時間によって決まります。

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed");
} else {
    gotoAndStop("waitForConnection");
}
```

関連項目

[connect \(XMLSocket.connect メソッド\)](#), [function ステートメント](#)

onData (XMLSocket.onData ハンドラ)

```
onData = function(src:String) {}
```

ゼロ (0) バイトで終了する XML メッセージがサーバーからダウンロードされると、呼び出されます。`XMLSocket.onData` イベントハンドラをオーバーライド (上書き定義) して、XML テキストを XML として解析せずに取得することができます。この機能は、転送するデータパケットのフォーマットが一定でない場合に、受け取ったデータを Flash Player で XML として解析するのではなく、直接操作するのに適しています。

デフォルトでは、`XMLSocket.onData` メソッドは `XMLSocket.onXML` メソッドを呼び出します。`XMLSocket.onData` を無効にしてカスタムビヘイビアを使用すると、`XMLSocket.onXML` は呼び出されなくなります。`XMLSocket.onXML` は `XMLSocket.onData` で呼び出す必要があります。

パラメータ

src:String - サーバーから送られたデータを含むストリング。

例

この例では、src パラメータはサーバーからダウンロードされた XML テキストを含むストリングです。ゼロ (0) バイトのターミネータは、ストリングに含まれません。

```
XMLSocket.prototype.onData = function (src) {  
    this.onXML(new XML(src));  
}
```

onXML (XMLSocket.onXML ハンドラ)

onXML = function(src:XML) {}

XML ドキュメントを含む指定の XML オブジェクトが、開いている XMLSocket 接続を通して届いたときに、Flash Lite Player によって呼び出されます。XMLSocket 接続を使用してクライアントとサーバーの間で転送できる XML ドキュメントの数に制限はありません。各ドキュメントは、値ゼロ (0) のバイトで終了します。Flash Lite Player がゼロバイトを受信すると、直前のゼロバイト以降に受信されたか、またはこれが最初に受信されたメッセージである場合は接続が確立された後に受信された、すべての XML を解析します。解析された XML の集合は1つの XML ドキュメントとして処理され、onXML() メソッドに渡されます。

このメソッドのデフォルトの実行形態では、アクションを実行しません。デフォルトの実行形態を無効にするには、独自に定義したアクションを含む関数を割り当てます。

パラメータ

src:XML - サーバーから受信した解析済み XML ドキュメントを含む XML オブジェクト。

例

次の例では、簡単なチャットアプリケーションで onXML() メソッドのデフォルトの実装を無効にします。関数 myOnXML() は、次の形式で1つの XML エレメント MESSAGE を認識するようにチャットアプリケーションに指示します。

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.  
var socket:XMLSocket = new XMLSocket();
```

次の displayMessage() 関数は、ユーザーが受信したメッセージを表示するユーザー定義関数であるとします。

```
socket.onXML = function (doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MESSAGE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

関連項目

[function ステートメント](#)

send (XMLSocket.send メソッド)

public send(data:[Object](#)) : Void

object パラメータで指定された XML オブジェクトまたはデータをストリングに変換し、その後ろにゼロ (0) バイトを附加してサーバーに転送します。object が XML オブジェクトである場合、ストリングは XML オブジェクトの XML テキスト表現です。送信操作は非同期です。つまり、転送処理はただちに終了しますが、データが転送されるのは、その後です。XMLSocket.send() メソッドは、データが正常に転送されたかどうかを示す値を返しません。

XMLSocket オブジェクトが XMLSocket.connect() メソッドを使ってサーバーに接続されていない場合、XMLSocket.send() 操作は失敗します。

パラメータ

data:[Object](#) - サーバーに転送する XML オブジェクトまたは他のデータ。

例

次の例では、XML オブジェクト my_xml をサーバーに送るためにユーザー名とパスワードを指定する方法を示します。

```
var myXMLSocket:XMLSocket = new XMLSocket();
var my_xml:XML = new XML();
var myLogin:XMLNode = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

関連項目

[connect \(XMLSocket.connect メソッド\)](#)

XMLSocket() コンストラクタ

public XMLSocket()

新しい XMLSocket オブジェクトを作成します。初期状態では、XMLSocket オブジェクトはサーバーに接続されません。オブジェクトをサーバーに接続するには、XMLSocket.connect() メソッドを呼び出す必要があります。

例

次の例では、XMLSocket オブジェクトを作成します。

```
var socket:XMLSocket = new XMLSocket();
```


使用されなくなった ActionScript

ActionScript の進化に伴い、使用されなくなった言語エレメントが数多く存在します。このセクションでは、使用を避ける項目を示し、代わりに使用できる言語エレメントを紹介します。使用されなくなったエレメントは Flash Lite 2.0 でまだ機能しますが、これらのエレメントをコードで使用しないことをお勧めします。使用されなくなったエレメントのサポートは今後、保証されません。

使用されなくなった関数の一覧

オプション	関数名	説明
	call(frame:Object)	非推奨 Flash Player 5 以降では使用しないでください。このアクションの代わりに function ステートメントを使用します。
	chr(number:Number) String	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに String.fromCharCode() を使用します。
	getProperty (my_mc:Object, property:Object)Object	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに Flash Player 5 で実装されたドットシンタックスを使用します。
	ifFrameLoaded ([scene:String], frame:Object, statement(s):Object)	非推奨 Flash Player 5 以降では使用しないでください。この関数は使用されなくなりました。MovieClip._framesloaded プロパティを使用することをお勧めします。
	int(value:Number) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに Math.round() を使用します。
	length (expression:String, variable:Object) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数およびすべてのストリング関数は使用されなくなりました。String クラスのメソッドと String.length プロパティを使用して同じ処理を行うことをお勧めします。

オプション	関数名	説明
	mbchr (number:Number)	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.fromCharCode()</code> メソッドを使用します。
	mblength(string:String) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.length</code> プロパティを使用します。
	mbord (character:String) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.charCodeAt()</code> メソッドを使用します。
	mbsubstring (value:String, index:Number, count:Number)String	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.substr()</code> メソッドを使用します。
	ord(character:String) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String</code> クラスのメソッドとプロパティを使用します。
	random(value:Number) Number	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>Math.random()</code> を使用します。
	substring(string:String, index:Number, count:Number)String	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>String.substr()</code> を使用します。
	tellTarget(target:String, statement(s):Object)	非推奨 Flash Player 5 以降では使用しないでください。代わりにドット(.)表記と <code>with</code> ステートメントを使用することをお勧めします。
	toggleHighQuality()	非推奨 Flash Player 5 以降では使用しないでください。この関数の代わりに <code>_quality</code> を使用します。

使用されなくなったプロパティの一覧

オプション	プロパティ名	説明
	\$version	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.version プロパティを使用します。
	_cap4WayKeyAS	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.has4WayKeyAS プロパティを使用します。
	_capCompoundSound	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasCompoundSound プロパティを使用します。
	_capEmail	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasEmail プロパティを使用します。
	_capLoadData	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasDataLoading プロパティを使用します。
	_capMFi	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasMFi プロパティを使用します。
	_capMIDI	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasMIDI プロパティを使用します。
	_capMMS	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasMMS プロパティを使用します。
	_capSMAF	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasSMAF プロパティを使用します。

オプション	プロパティ名	説明
	_capSMS	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasSMS プロパティを使用します。
	_capStreamSound	非推奨 Flash Lite 2.0 プレーヤー以降では使用しないでください。このアクションの代わりに System.capabilities.hasStreamingAudio プロパティを使用します。
	Button._highquality	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに Button._quality を使用します。
	MovieClip._highquality	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに MovieClip._quality を使用します。
	TextField._highquality	非推奨 Flash Player 7 以降では使用しないでください。このプロパティの代わりに TextField._quality を使用します。
	_highquality	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに _quality を使用します。
	maxscroll	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに TextField.maxscroll を使用します。
	scroll	非推奨 Flash Player 5 以降では使用しないでください。このプロパティの代わりに TextField.scroll を使用します。

使用されなくなった演算子の一覧

演算子	説明
<code>◊(不等価)</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子は使用されなくなりました。 <code>!= (inequality)</code> 演算子を使用することをお勧めします。
<code>add (concatenation (strings))</code>	非推奨 Flash Player 5 以降では使用しないでください。Flash Player 5 以降用のコンテンツを作成する場合は、加算 (+) 演算子を使用することをお勧めします。 メモ : Flash Lite 2.0 では、 <code>add</code> 演算子も使用されなくなりました。代わりに加算 (+) 演算子を使用します。
<code>and (論理積 (AND))</code>	非推奨 Flash Player 5 以降では使用しないでください。論理積 (&&) 演算子を使用することをお勧めします。
<code>eq (等価 (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>== (等価)</code> 演算子を使用します。
<code>ge (大きい、または等しい (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>>= (より大きい、または等しい)</code> 演算子を使用します。
<code>gt (より大きい (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>> (より大きい)</code> 演算子を使用します。
<code>le (小さい、または等しい (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code><= (より小さい、または等しい)</code> 演算子を使用します。
<code>lt (より小さい (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>< (より小さい)</code> 演算子を使用します。
<code>ne (不等価 (ストリング))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>!= (inequality)</code> 演算子を使用します。
<code>not (論理否定 (NOT))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code>! (logical NOT)</code> 演算子を使用します。
<code>or (論理和 (OR))</code>	非推奨 Flash Player 5 以降では使用しないでください。この演算子の代わりに <code> (論理和)</code> 演算子を使用します。

第4章

サポートされていない ActionScript

Flash Lite 2.1 は次のエレメントをサポートしていません。

サポートされていないクラス

Accessibility、Camera、ContextMenu、ContextMenuItem、CustomActions、
LocalConnection、Locale(mx.lang.Locale)、Microphone、NetConnection、NetStream、
PrintJob、TextField.StyleSheet、TextSnapshot、XMLUI

サポートされていないメソッド

Mouse.hide、Mouse.show、MovieClip.attachAudio、MovieClip.getTextSnapshot、
Selection.getBeginIndex、Selection.getCaretIndex、Selection.getEndIndex、
System.setClipboard、System.showSettings、TextField.getFontList、Video.attachVideo、
Video.clear

サポートされていないプロパティ

```
Button.blendMode、Button.cacheAsBitmap、Button.filters、Button.menu、  
Button.useHandCursor、System.capabilities.language、System.capabilities.manufacturer、  
System.capabilities.pixelAspectRatio、System.capabilities.playerType、  
System.capabilities.screenColor、System.capabilities.screenDPI、  
System.capabilities.serverString、Key.isToggled、MovieClip.menu、  
MovieClip.useHandCursor、Stage.showMenu、System.exactSettings、TextField.menu、  
TextField.mouseWheelEnabled、TextField.restrict、Video._alpha、Video.deblocking、  
Video._height、Video.height、Video._name、Video._parent、Video._rotation、  
Video.smoothing、Video._visible、Video._width、Video.width、Video._x、Video._xmouse、  
Video._xscale、Video._y、Video._ymouse、Video._yscale
```

サポートされていないグローバル関数

```
asfunction、MMExecute、print、printAsBitmap、printAsBitmapNum、printNum、  
updateAfterEvent
```

サポートされていないイベントハンドラ

```
onUpdate、Mouse.onMouseWheel
```

サポートされていない fscommand

```
allowscale、exec、fullscreen、quit、showmenu、trapallkeys
```

索引

記号

〈より小さい演算子 129
≤より小さいか等しい演算子 130
« ビット単位の左シフト演算子 103
≪ビット単位の左シフト後代入演算子 104
◇不等価演算子 128
!論理否定 (NOT) 演算子 133
!=不等価演算子 126
!=厳密な不等価演算子 145
"ストリング区切り記号演算子 146
#endinitclip ディレクティブ 7
#include ディレクティブ 8
#initclip ディレクティブ 9
\$version プロパティ 75
% 剰余演算子 137
%= 剰余代入演算子 137
& ビット単位の論理積 (AND) 演算子 102
&& 論理積 (AND) 演算子 132
&= ビット単位の論理積 (AND) 代入演算子 103
() 括弧演算子 142
* 乗算演算子 138
*= 乗算後代入演算子 139
+ 加算演算子 96
++ インクリメント演算子 124
+= 加算後代入演算子 97
, カンマ演算子 114
- 減算演算子 146
-- デクリメント演算子 117
-= 減算後代入演算子 147
-Infinity 定数 12
. ドット演算子 119
/ 除算演算子 118
/* コメントブロック区切り記号演算子 113
// コメント行区切り記号演算子 131
/= 除算後代入演算子 118
: タイプ演算子 148
= 代入演算子 100

== 等価演算子 120
=== 厳密な等価演算子 143
>より大きい演算子 122
>=より大きいか等しい演算子 123
» ビット単位の右シフト演算子 108
»= ビット単位の右シフト後代入演算子 109
»» ビット単位の符号なし右シフト演算子 110
»»= ビット単位の符号なし右シフト後代入演算子 111
? 条件演算子 116
[] 配列アクセス演算子 98
^ ビット単位の排他的論理和 (XOR) 演算子 112
^= ビット単位の排他的論理和 (XOR) 代入演算子 113
proto プロパティ 497
_resolve プロパティ 500
_alpha プロパティ 240, 393, 588
_cap4WayKeyAS プロパティ 76
_capCompoundSound プロパティ 77
_capEmail プロパティ 77
_capLoadData プロパティ 78
_capMFi プロパティ 78
_capMIDI プロパティ 79
_capMMS プロパティ 80
_capSMAF プロパティ 80
_capSMS プロパティ 81
_capStreamSound プロパティ 82
_currentframe プロパティ 402
_droptarget プロパティ 405
_focusrect プロパティ 82, 241, 409
_forceframerate プロパティ 83
_framesloaded プロパティ 410
_global プロパティ 83
_height プロパティ 243, 423, 597
_highquality プロパティ 84, 244, 423, 597
_level プロパティ 85
_listeners プロパティ 339
_lockroot プロパティ 433
_name プロパティ 244, 436, 601

_parent プロパティ 86, 251, 449, 605
_quality プロパティ 87, 252, 451, 607
_root プロパティ 87
_rotation プロパティ 252, 453, 611
_soundbuftime プロパティ 89, 253, 454, 616
_target プロパティ 255, 461, 618
_totalframes プロパティ 461
_url プロパティ 257, 463, 621
_visible プロパティ 257, 464, 622
_width プロパティ 258, 464, 622
_x プロパティ 258, 465, 624
_xmouse プロパティ 259, 465, 624
_xscale プロパティ 260, 466, 625
_y プロパティ 260, 467, 626
_ymouse プロパティ 261, 467, 626
_yscale プロパティ 261, 468, 627
{} オブジェクト初期化演算子 141
| ビット単位の論理和 (OR) 演算子 106
|= ビット単位の排他的論理和 (OR) 代入演算子 107
|| 論理和 (OR) 演算子 135
- ビット単位の否定 (NOT) 演算子 105

A

abs() メソッド 363
acos() メソッド 364
addListener() メソッド 330, 379, 471, 515, 526, 558, 586
addProperty() メソッド 491
addRequestHeader() メソッド 348, 651
align プロパティ 559, 630
allowDomain() メソッド 509
allowInsecureDomain() メソッド 510
and 論理積 (AND) 演算子 133
appendChild() メソッド 670
apply() メソッド 325
arguments
 callee プロパティ 214
 caller プロパティ 214
 length プロパティ 215
arguments クラス 213
Array
 Array() コンストラクタ 218
 CASEINSENSITIVE プロパティ 219
 concat() メソッド 220
 DESCENDING プロパティ 221
 join() メソッド 221
 length プロパティ 222
 NUMERIC プロパティ 223
 pop() メソッド 223

push() メソッド 224
RETURNINDEXEDARRAY プロパティ 224
reverse() メソッド 225
shift() メソッド 225
slice() メソッド 226
sort() メソッド 227
sortOn() メソッド 229
splice() メソッド 232
toString() メソッド 233
UNIQUESORT プロパティ 234
unshift() メソッド 234
Array 関数 20
Array クラス 215
Array() コンストラクタ 218
asin() メソッド 364
atan() メソッド 365
atan2() メソッド 366
attachMovie() メソッド 394
attachSound() メソッド 537
attributes プロパティ 671
audioMIMETypes プロパティ 266
autoSize プロパティ 588
avHardwareDisable プロパティ 267

B

background プロパティ 590
backgroundColor プロパティ 591
BACKSPACE プロパティ 331
beginFill() メソッド 395
beginGradientFill() メソッド 396
blockIndent プロパティ 630
bold プロパティ 631
Boolean
 Boolean() コンストラクタ 236
 toString() メソッド 236
 valueOf() メソッド 237
Boolean 関数 22
Boolean クラス 235
Boolean() コンストラクタ 236
border プロパティ 591
borderColor プロパティ 591
bottomScroll プロパティ 592
break ステートメント 152
bullet プロパティ 631
Button
 _alpha プロパティ 240
 _focusrect プロパティ 241
 _height プロパティ 243
 _highquality プロパティ 244

_name プロパティ 244
_parent プロパティ 251
_quality プロパティ 252
_rotation プロパティ 252
_soundbuftime プロパティ 253
_target プロパティ 255
_url プロパティ 257
_visible プロパティ 257
_width プロパティ 258
_x プロパティ 258
_xmouse プロパティ 259
_xscale プロパティ 260
_y プロパティ 260
_ymouse プロパティ 261
_yscale プロパティ 261
enabled プロパティ 241
getDepth() メソッド 242
tabEnabled プロパティ 253
tablIndex プロパティ 254
trackAsMenu プロパティ 256

Button クラス 237
onDragOut イベント 245
onDragOver イベント 245
onKeyDown イベント 246
onKeyUp イベント 247
onKillFocus イベント 248
onPress イベント 248
onRelease イベント 249
onReleaseOutside イベント 249
onRollOut イベント 250
onRollOver イベント 250
onSetFocus イベント 250

C

call 関数 23
call() メソッド 327
callee プロパティ 214
caller プロパティ 214
capabilities
 audioMIMETypes プロパティ 266
 avHardwareDisable プロパティ 267
 has4WayKeyAS プロパティ 267
 hasAccessibility プロパティ 267
 hasAudio プロパティ 268
 hasAudioEncoder プロパティ 268
 hasCMIDI プロパティ 268
 hasCompoundSound プロパティ 269
 hasDataLoading プロパティ 269
 hasEmail プロパティ 270

hasEmbeddedVideo プロパティ 270
hasMappableSoftKeys プロパティ 270
hasMFI プロパティ 271
hasMIDI プロパティ 271
hasMMS プロパティ 271
hasMouse プロパティ 272
hasMP3 プロパティ 272
hasPrinting プロパティ 272
hasQWERTYKeyboard プロパティ 273
hasScreenBroadcast プロパティ 273
hasScreenPlayback プロパティ 273
hasSharedObjects プロパティ 274
hasSMAF プロパティ 274
hasSMS プロパティ 274
hasStreamingAudio プロパティ 275
hasStreamingVideo プロパティ 275
hasStylus プロパティ 275
hasVideoEncoder プロパティ 276
hasXMLSocket プロパティ 276
imageMIMETypes プロパティ 276
isDebugger プロパティ 277
language プロパティ 277
localFileReadDisable プロパティ 278
MIMETypes プロパティ 279
os プロパティ 279
screenOrientation プロパティ 279
screenResolutionX プロパティ 280
screenResolutionY プロパティ 280
softKeyCount プロパティ 280
version プロパティ 281
videoMIMETypes プロパティ 281
capabilities クラス 262
CAPSLOCK プロパティ 331
case ステートメント 153
CASEINSENSITIVE プロパティ 219
ceil() メソッド 366
charAt() メソッド 566
charCodeAt() メソッド 567
childNodes プロパティ 672
chr 関数 24
class
 LoadVars により送り出す 355、356
 MovieClip により送り出す 437、439、440、441、
 442、443、444、445、446、447、448
 MovieClipLoader により送り出す 476、477、479、
 480、481
 Selection により送り出す 517
 SharedObject により送り出す 532
 Sound により送り出す 548、549、550
 Stage により送り出す 560

System により送り出す 579
TextField により送り出す 602、603、605
Video により送り出す 645
XML により送り出す 660、661
XMLSocket により送り出す 690、691、692
キーにより送り出す 340
マウスにより送り出す 381、382、383
ボタンにより送り出す 245、246、247、248、249、
250
class ステートメント 154
clear() メソッド 399、526
clearInterval 関数 24
cloneNode() メソッド 673
close() メソッド 644、688
Color
 Color() コンストラクタ 283
 getRGB() メソッド 283
 getTransform() メソッド 284
 setRGB() メソッド 284
 setTransform() メソッド 285
Color クラス 281
color プロパティ 632
Color() コンストラクタ 283
concat() メソッド 220、567
condenseWhite プロパティ 592
connect() メソッド 688
constructor プロパティ 494
contentType プロパティ 349、652
continue ステートメント 156
CONTROL プロパティ 331
cos() メソッド 367
createElement() メソッド 652
createEmptyMovieClip() メソッド 399
createTextField() メソッド 400
createTextNode() メソッド 653
curveTo() メソッド 403

D

data プロパティ 527
Date
 Date() コンストラクタ 291
 getDate() メソッド 293
 getDay() メソッド 293
 getFullYear() メソッド 294
 getHours() メソッド 294
 getLocaleLongDate() メソッド 295
 getLocaleShortDate() メソッド 296
 getLocaleTime() メソッド 296
 getMilliseconds() メソッド 297
 getMinutes() メソッド 297
 getMonth() メソッド 297
 getSeconds() メソッド 298
 getTime() メソッド 298
 getTimezoneOffset() メソッド 299
 getUTCDate() メソッド 299
 getUTCDay() メソッド 300
 getUTCFullYear() メソッド 300
 getUTCHours() メソッド 301
 getUTCMilliseconds() メソッド 301
 getUTCMinutes() メソッド 302
 getUTCMonth() メソッド 302
 getUTCSeconds() メソッド 303
 getUTCYear() メソッド 303
 getYear() メソッド 304
 setDate() メソッド 304
 setFullYear() メソッド 305
 setHours() メソッド 306
 setMilliseconds() メソッド 306
 setMinutes() メソッド 307
 setMonth() メソッド 307
 setSeconds() メソッド 308
 setTime() メソッド 308
 setUTCDate() メソッド 309
 setUTCFullYear() メソッド 309
 setUTCHours() メソッド 310
 setUTCMilliseconds() メソッド 311
 setUTCMinutes() メソッド 312
 setUTCMonth() メソッド 312
 setUTCSeconds() メソッド 313
 setYear() メソッド 314
 toString() メソッド 314
 UTC() メソッド 315
 valueOf() メソッド 315
Date クラス 286
Date() コンストラクタ 291
decode() メソッド 349
default ステートメント 157
delete ステートメント 158
DELETEKEY プロパティ 331
DESCENDING プロパティ 221
do..while ステートメント 160
docTypeDecl プロパティ 654
DOWN プロパティ 332
duplicateMovieClip 関数 25
duplicateMovieClip() メソッド 406
duration プロパティ 537
dynamic ステートメント 161

E

E プロパティ 367
else if ステートメント 163
else ステートメント 162
embedFonts プロパティ 593
enabled プロパティ 241, 407
END プロパティ 333
endFill() メソッド 408
ENTER プロパティ 333
eq 等値(ストリング)演算子 122
Error
 Error() コンストラクタ 317
 message プロパティ 318
 name プロパティ 319
 toString() メソッド 320
Error クラス 316
Error() コンストラクタ 317
escape 関数 26
ESCAPE プロパティ 334
eval 関数 27
exp() メソッド 368
ExtendBacklightDuration コマンド 195
ExtendedKey
 SOFT1 プロパティ 322
 SOFT10 プロパティ 323
 SOFT11 プロパティ 323
 SOFT12 プロパティ 323
 SOFT2 プロパティ 323
 SOFT3 プロパティ 323
 SOFT4 プロパティ 323
 SOFT5 プロパティ 323
 SOFT6 プロパティ 324
 SOFT7 プロパティ 324
 SOFT8 プロパティ 324
 SOFT9 プロパティ 324
ExtendedKey クラス 320
extends ステートメント 163

F

false 定数 11
firstChild プロパティ 674
floor() メソッド 368
flush() メソッド 529
focusEnabled プロパティ 409
font プロパティ 632
for ステートメント 166
for..in ステートメント 167
fromCharCode() メソッド 568

fscommand 関数 28
fscommand2 関数 30
fscommand2 コマンド 194
FullScreen コマンド 196
Function
 apply() メソッド 325
 call() メソッド 327
Function クラス 324
function ステートメント 168

G

ge 大きい、または等しい(ストリング)演算子 124
get ステートメント 170
getAscii() メソッド 335
GetBatteryLevel コマンド 196
getBounds() メソッド 411
getBytesLoaded() メソッド 350, 412, 539, 655
getBytesTotal() メソッド 351, 413, 541, 656
getCode() メソッド 336
getDate() メソッド 293
getDay() メソッド 293
getDepth() メソッド 242, 414, 594
GetDevice コマンド 197
GetDeviceID コマンド 197
getFocus() メソッド 516
GetFreePlayerMemory コマンド 198
getFullYear() メソッド 294
getHours() メソッド 294
getInstanceAtDepth() メソッド 414
getLocal() メソッド 530
getLocaleLongDate() メソッド 295
getLocaleShortDate() メソッド 296
getLocaleTime() メソッド 296
GetMaxBatteryLevel コマンド 198
GetMaxSignalLevel コマンド 198
getMaxSize() メソッド 531
GetMaxVolumeLevel コマンド 199
getMilliseconds() メソッド 297
getMinutes() メソッド 297
getMonth() メソッド 297
GetNetworkConnectionName コマンド 199
GetNetworkConnectStatus コマンド 200
GetNetworkGeneration コマンド 201
GetNetworkName コマンド 201
GetNetworkRequestStatus コマンド 202
GetNetworkStatus コマンド 203
getNewTextFormat() メソッド 595
getNextHighestDepth() メソッド 415
getPan() メソッド 541

GetPlatform コマンド 204
GetPowerSource コマンド 205
getProgress() メソッド 472
getProperty 関数 31
getRGB() メソッド 283
getSeconds() メソッド 298
GetSignalLevel コマンド 205
getSize() メソッド 532
GetSoftKeyLocation コマンドト 206
getSWFVersion() メソッド 417
getTextExtent() メソッド 633
getTextFormat() メソッド 595
getTime() メソッド 298
getTimer 関数 32
getTimezoneOffset() メソッド 299
GetTotalPlayerMemory コマンド 206
getTransform() メソッド 284、542
getUrl 関数 32
getUrl() メソッド 417
getUTCDate() メソッド 299
getUTCDay() メソッド 300
getUTCFullYear() メソッド 300
getUTCHours() メソッド 301
getUTCMilliseconds() メソッド 301
getUTCMinutes() メソッド 302
getUTCMonth() メソッド 302
getUTCSeconds() メソッド 303
getUTCYear() メソッド 303
getVersion 関数 34
getVolume() メソッド 544
GetVolumeLevel コマンド 207
getYear() メソッド 304
globalToLocal() メソッド 419
gotoAndPlay 関数 34
gotoAndPlay() メソッド 421
gotoAndStop 関数 35
gotoAndStop() メソッド 422
gt より大きい(ストリング)演算子 123

H

has4WayKeyAS プロパティ 267
hasAccessibility プロパティ 267
hasAudio プロパティ 268
hasAudioEncoder プロパティ 268
hasChildNodes() メソッド 676
hasCMIDI プロパティ 268
hasCompoundSound プロパティ 269
hasDataLoading プロパティ 269
hasEmail プロパティ 270

hasEmbeddedVideo プロパティ 270
hasMappableSoftKeys プロパティ 270
hasMFI プロパティ 271
hasMIDI プロパティ 271
hasMMS プロパティ 271
hasMouse プロパティ 272
hasMP3 プロパティ 272
hasOwnProperty() メソッド 495
hasPrinting プロパティ 272
hasQWERTYKeyboard プロパティ 273
hasScreenBroadcast プロパティ 273
hasScreenPlayback プロパティ 273
hasSharedObjects プロパティ 274
hasSMAF プロパティ 274
hasSMS プロパティ 274
hasStreamingAudio プロパティ 275
hasStreamingVideo プロパティ 275
hasStylus プロパティ 275
hasVideoEncoder プロパティ 276
hasXMLSocket プロパティ 276
height プロパティ 560
hitArea プロパティ 424
hitTest() メソッド 424
HOME プロパティ 337
hscroll プロパティ 597
html プロパティ 598
htmlText プロパティ 599

I

id3 プロパティ 545
if ステートメント 171
ifFrameLoaded 関数 36
ignoreWhite プロパティ 656
imageMIMETypes プロパティ 276
implements ステートメント 172
import ステートメント 172
indent プロパティ 635
indexOf() メソッド 568
Infinity 定数 12
INSERT プロパティ 337
insertBefore() メソッド 676
instanceof 演算子 128
int 関数 37
interface ステートメント 173
intrinsic ステートメント 175
isDebugger プロパティ 277
isDown() メソッド 338
isFinite 関数 37
isNaN 関数 38

isPropertyEnumerable() メソッド 495
isPrototypeOf() メソッド 496
italic プロパティ 636

J

join() メソッド 221

K

Key

_listeners プロパティ 339
addListener() メソッド 330
BACKSPACE プロパティ 331
CAPSLOCK プロパティ 331
CONTROL プロパティ 331
DELETEKEY プロパティ 331
DOWN プロパティ 332
END プロパティ 333
ENTER プロパティ 333
ESCAPE プロパティ 334
getAscii() メソッド 335
getCode() メソッド 336
HOME プロパティ 337
INSERT プロパティ 337
isDown() メソッド 338
LEFT プロパティ 338
PGDN プロパティ 341
PGUP プロパティ 341
removeListener() メソッド 342
RIGHT プロパティ 342
SHIFT プロパティ 343
SPACE プロパティ 344
TAB プロパティ 344
UP プロパティ 345
Key クラス 328
onKeyDown イベント 340
onKeyUp イベント 340

L

language プロパティ 277
lastChild プロパティ 677
lastIndexOf() メソッド 569
le 小さい、または等しい(ストリング)演算子 131
leading プロパティ 636
LEFT プロパティ 338
leftMargin プロパティ 637
length 関数 39

length プロパティ 215, 222, 570, 599
lineStyle() メソッド 425
lineTo() メソッド 427
LN10 プロパティ 369
LN2 プロパティ 369
load() メソッド 352, 658
loadClip() メソッド 474
loaded プロパティ 354, 659
loadMovie 関数 39
loadMovie() メソッド 428
loadMovieNum 関数 41
loadPolicyFile() メソッド 512
loadSound() メソッド 548
loadVariables 関数 42
loadVariables() メソッド 429
loadVariablesNum 関数 44
LoadVars
 addRequestHeader() メソッド 348
 contentType プロパティ 349
 decode() メソッド 349
 getBytesLoaded() メソッド 350
 getBytesTotal() メソッド 351
 load() メソッド 352
 loaded プロパティ 354
 LoadVars() コンストラクタ 354
 send() メソッド 357
 sendAndLoad() メソッド 358
 toString() メソッド 360
LoadVars クラス 346
 onData イベント 355
 onLoad イベント 356
LoadVars() コンストラクタ 354
localFileReadDisable プロパティ 278
localToGlobal() メソッド 431
log() メソッド 369
LOG10E プロパティ 370
LOG2E プロパティ 370
lt より小さい(ストリング)演算子 130

M

Math

abs() メソッド 363
acos() メソッド 364
asin() メソッド 364
atan() メソッド 365
atan2() メソッド 366
ceil() メソッド 366
cos() メソッド 367
E プロパティ 367

`exp()` メソッド 368
`floor()` メソッド 368
`LN10` プロパティ 369
`LN2` プロパティ 369
`log()` メソッド 369
`LOG10E` プロパティ 370
`LOG2E` プロパティ 370
`max()` メソッド 371
`min()` メソッド 371
`PI` プロパティ 372
`pow()` メソッド 373
`random()` メソッド 374
`round()` メソッド 374
`sin()` メソッド 375
`sqrt()` メソッド 376
`SQRT1_2` プロパティ 377
`SQRT2` プロパティ 377
`tan()` メソッド 377
Math クラス 361
`max()` メソッド 371
`MAX_VALUE` プロパティ 486
`maxChars` プロパティ 599
`maxhscroll` プロパティ 600
`maxscroll` プロパティ 86、600
`mbchr` 関数 46
`mblength` 関数 46
`mbord` 関数 47
`mbsubstr` 関数 47
`message` プロパティ 318
`MIMETypes` プロパティ 279
`min()` メソッド 371
`MIN_VALUE` プロパティ 487
Mouse
 `addListener()` メソッド 379
 `removeListener()` メソッド 384
Mouse クラス 378
 `onMouseDown` イベント 381
 `onMouseMove` イベント 382
 `onMouseUp` イベント 383
`moveTo()` メソッド 436
MovieClip
 `_alpha` プロパティ 393
 `_currentframe` プロパティ 402
 `_droptarget` プロパティ 405
 `_focusrect` プロパティ 409
 `_framesloaded` プロパティ 410
 `_height` プロパティ 423
 `_highquality` プロパティ 423
 `_lockroot` プロパティ 433
 `_name` プロパティ 436
 `_parent` プロパティ 449
 `_quality` プロパティ 451
 `_rotation` プロパティ 453
 `_soundbuftime` プロパティ 454
 `_target` プロパティ 461
 `_totalframes` プロパティ 461
 `_url` プロパティ 463
 `_visible` プロパティ 464
 `_width` プロパティ 464
 `_x` プロパティ 465
 `_xmouse` プロパティ 465
 `_xscale` プロパティ 466
 `_y` プロパティ 467
 `_ymouse` プロパティ 467
 `_yscale` プロパティ 468
 `attachMovie()` メソッド 394
 `beginFill()` メソッド 395
 `beginGradientFill()` メソッド 396
 `clear()` メソッド 399
 `createEmptyMovieClip()` メソッド 399
 `createTextField()` メソッド 400
 `curveTo()` メソッド 403
 `duplicateMovieClip()` メソッド 406
 `enabled` プロパティ 407
 `endFill()` メソッド 408
 `focusEnabled` プロパティ 409
 `getBounds()` メソッド 411
 `getBytesLoaded()` メソッド 412
 `getBytesTotal()` メソッド 413
 `getDepth()` メソッド 414
 `getInstanceAtDepth()` メソッド 414
 `getNextHighestDepth()` メソッド 415
 `getSWFVersion()` メソッド 417
 `getURL()` メソッド 417
 `globalToLocal()` メソッド 419
 `gotoAndPlay()` メソッド 421
 `gotoAndStop()` メソッド 422
 `hitArea` プロパティ 424
 `hitTest()` メソッド 424
 `lineStyle()` メソッド 425
 `lineTo()` メソッド 427
 `loadMovie()` メソッド 428
 `loadVariables()` メソッド 429
 `localToGlobal()` メソッド 431
 `moveTo()` メソッド 436
 `nextFrame()` メソッド 437
 `play()` メソッド 449
 `prevFrame()` メソッド 450
 `removeMovieClip()` メソッド 452
 `setMask()` メソッド 454

startDrag() メソッド 455
stop() メソッド 456
stopDrag() メソッド 456
swapDepths() メソッド 457
tabChildren プロパティ 458
tabEnabled プロパティ 459
tabIndex プロパティ 460
trackAsMenu プロパティ 462
unloadMovie() メソッド 462

MovieClip クラス 385
onData イベント 437
onDragOut イベント 439
onDragOver イベント 439
onEnterFrame イベント 440
onKeyDown イベント 440
onKeyUp イベント 441
onKillFocus イベント 442
onLoad イベント 443
onMouseDown イベント 444
onMouseMove イベント 444
onMouseUp イベント 445
onPress イベント 445
onRelease イベント 446
onReleaseOutside イベント 446
onRollOut イベント 447
onRollOver イベント 447
onSetFocus イベント 447
onUpload イベント 448

MovieClipLoader
addListener() メソッド 471
getProgress() メソッド 472
loadClip() メソッド 474
MovieClipLoader() コンストラクタ 476
removeListener() メソッド 482
unloadClip() メソッド 484

MovieClipLoader クラス 469
onLoadComplete イベント 476
onLoadError イベント 477
onLoadInit イベント 479
onLoadProgress イベント 480
onLoadStart イベント 481

MovieClipLoader() コンストラクタ 476
multiline プロパティ 601

N

name プロパティ 319
NaN 定数 12
NaN プロパティ 487
ne 不等値(ストリング)演算子 140

NEGATIVE_INFINITY プロパティ 487
new 演算子 139
newline 定数 12
nextFrame 関数 48
nextFrame() メソッド 437
nextScene 関数 48
nextSibling プロパティ 678
nodeName プロパティ 679
nodeType プロパティ 680
nodeValue プロパティ 681
not 論理否定(NOT)演算子 134
null 定数 13
Number
MAX_VALUE プロパティ 486
MIN_VALUE プロパティ 487
NaN プロパティ 487
NEGATIVE_INFINITY プロパティ 487
Number() コンストラクタ 488
POSITIVE_INFINITY プロパティ 488
toString() メソッド 489
valueOf() メソッド 489

Number 関数 49
Number クラス 485
Number() コンストラクタ 488
NUMERIC プロパティ 223

O

Object
__proto__ プロパティ 497
_resolve プロパティ 500
addProperty() メソッド 491
constructor プロパティ 494
hasOwnProperty() メソッド 495
isPrototypeOf() メソッド 495
isPrototypeOf() メソッド 496
Object() コンストラクタ 497
prototype プロパティ 498
registerClass() メソッド 499
toString() メソッド 503
unwatch() メソッド 504
valueOf() メソッド 505
watch() メソッド 506

Object 関数 50
Object クラス 490
Object() コンストラクタ 497
on ハンドラ 51
onChanged イベント 602
onClipEvent ハンドラ 52
onClose イベント 690

onConnect イベント 690
onData イベント 355、437、660、691
onDragOut イベント 245、439
onDragOver イベント 245、439
onEnterFrame イベント 440
onID3 イベント 548
onKeyDown イベント 246、340、440
onKeyUp イベント 247、340、441
onKillFocus イベント 248、442、602
onLoad イベント 356、443、549、661
onLoadComplete イベント 476
onLoadError イベント 477
onLoadInit イベント 479
onLoadProgress イベント 480
onLoadStart イベント 481
onMouseDown イベント 381、444
onMouseMove イベント 382、444
onMouseUp イベント 383、445
onPress イベント 248、445
onRelease イベント 249、446
onReleaseOutside イベント 249、446
onResize イベント 560
onRollOut イベント 250、447
onRollOver イベント 250、447
onScroller イベント 603
onSetFocus イベント 250、447、517、605
onSoundComplete イベント 550
onStatus イベント 532、579、645
onUnload イベント 448
onXML イベント 692
or 論理和 (OR) 演算子 136
ord 関数 54
os プロパティ 279

P

parentNode プロパティ 683
parseFloat 関数 54
parseInt 関数 55
parseXML() メソッド 662
password プロパティ 606
pause() メソッド 645
PGDN プロパティ 341
PGUP プロパティ 341
PI プロパティ 372
play() メソッド 449、646
play 関数 56
pop() メソッド 223
position プロパティ 551
POSITIVE_INFINITY プロパティ 488

pow() メソッド 373
prevFrame 関数 57
prevFrame() メソッド 450
previousSibling プロパティ 683
prevScene 関数 57
private ステートメント 177
prototype プロパティ 498
public ステートメント 178
push() メソッド 224

R

random 関数 57
random() メソッド 374
registerClass() メソッド 499
removeListener() メソッド 342、384、482、519、
534、561、607
removeMovieClip 関数 58
removeMovieClip() メソッド 452
removeNode() メソッド 684
removeTextField() メソッド 608
replaceSel() メソッド 609
replaceText() メソッド 610
ResetSoftKeys コマンド 208
resume() メソッド 646
return ステートメント 179
RETURNINDEXEDARRAY プロパティ 224
reverse() メソッド 225
RIGHT プロパティ 342
rightMargin プロパティ 637
round() メソッド 374

S

scaleMode プロパティ 562
screenOrientation プロパティ 279
screenResolutionX プロパティ 280
screenResolutionY プロパティ 280
scroll プロパティ 88、611
security
allowDomain() メソッド 509
allowInsecureDomain() メソッド 510
loadPolicyFile() メソッド 512
security クラス 508
selectable プロパティ 612
Selection
addListener() メソッド 515
getFocus() メソッド 516
removeListener() メソッド 519

setFocus() メソッド 520
setSelection() メソッド 521
Selection クラス 514
onSetFocus イベント 517
send() メソッド 357、662、693
sendAndLoad() メソッド 358、663
set variable ステートメント 181
set ステートメント 180
 setDate() メソッド 304
setFocus() メソッド 520
SetFocusRectColor コマンド 208
setFullYear() メソッド 305
setHours() メソッド 306
SetInputTextType コマンド 209
setInterval 関数 59
setMask() メソッド 454
setMilliseconds() メソッド 306
setMinutes() メソッド 307
setMonth() メソッド 307
setNewTextFormat() メソッド 613
setPan() メソッド 551
setProperty 関数 62
setRGB() メソッド 284
setSeconds() メソッド 308
setSelection() メソッド 521
SetSoftKeys コマンド 210
setTextFormat() メソッド 614
 setTime() メソッド 308
setTransform() メソッド 285、552
setUTCDate() メソッド 309
setUTCFullYear() メソッド 309
setUTCHours() メソッド 310
setUTCMilliseconds() メソッド 311
setUTCMinutes() メソッド 312
setUTCMonth() メソッド 312
setUTCSeconds() メソッド 313
setVolume() メソッド 554
setYear() メソッド 314
SharedObject
 addListener() メソッド 526
 clear() メソッド 526
 data プロパティ 527
 flush() メソッド 529
 getLocal() メソッド 530
 getMaxSize() メソッド 531
 getSize() メソッド 532
 removeListener() メソッド 534
SharedObject クラス 522
onStatus イベント 532
SHIFT プロパティ 343
shift() メソッド 225
sin() メソッド 375
size プロパティ 638
slice() メソッド 226、571
SOFT1 プロパティ 322
SOFT10 プロパティ 323
SOFT11 プロパティ 323
SOFT12 プロパティ 323
SOFT2 プロパティ 323
SOFT3 プロパティ 323
SOFT4 プロパティ 323
SOFT5 プロパティ 323
SOFT6 プロパティ 324
SOFT7 プロパティ 324
SOFT8 プロパティ 324
SOFT9 プロパティ 324
softKeyCount プロパティ 280
sort() メソッド 227
sortOn() メソッド 229
Sound
 attachSound() メソッド 537
 duration プロパティ 537
getBytesLoaded() メソッド 539
getBytesTotal() メソッド 541
getPan() メソッド 541
getTransform() メソッド 542
getVolume() メソッド 544
id3 プロパティ 545
loadSound() メソッド 548
position プロパティ 551
setPan() メソッド 551
setTransform() メソッド 552
setVolume() メソッド 554
Sound() コンストラクタ 554
start() メソッド 555
stop() メソッド 556
Sound クラス 534
 onID3 イベント 548
 onLoad イベント 549
 onSoundComplete イベント 550
Sound() コンストラクタ 554
SPACE プロパティ 344
splice() メソッド 232
split() メソッド 572
sqrt() メソッド 376
SQRT1_2 プロパティ 377
SQRT2 プロパティ 377

Stage
 addListener() メソッド 558
 align プロパティ 559
 height プロパティ 560
 removeListener() メソッド 561
 scaleMode プロパティ 562
 width プロパティ 563
Stage クラス 556
 onResize イベント 560
start() メソッド 555
startDrag 関数 62
startDrag() メソッド 455
StartVibrate コマンド 212
static ステートメント 182
status プロパティ 665
stop 関数 63
stop() メソッド 456、556、647
stopAllSounds 関数 64
stopDrag 関数 65
stopDrag() メソッド 456
StopVibrate コマンド 212
String
 charAt() メソッド 566
 charCodeAt() メソッド 567
 concat() メソッド 567
 fromCharCode() メソッド 568
 indexOf() メソッド 568
 lastIndexOf() メソッド 569
 length プロパティ 570
 slice() メソッド 571
 split() メソッド 572
 String() コンストラクタ 574
 substr() メソッド 574
 substring() メソッド 575
 toLowerCase() メソッド 576
 toString() メソッド 576
 toUpperCase() メソッド 577
 valueOf() メソッド 578
String 関数 65
String クラス 563
String() コンストラクタ 574
substr() メソッド 574
substring 関数 66
substring() メソッド 575
super ステートメント 183
swapDepths() メソッド 457
switch ステートメント 184
System
 useCodepage プロパティ 580

System クラス 578
 onStatus イベント 579

T

TAB プロパティ 344
tabChildren プロパティ 458
tabEnabled プロパティ 253、459、616
tabIndex プロパティ 254、460、617
tabStops プロパティ 638
tan() メソッド 377
target プロパティ 639
targetPath 関数 67
tellTarget 関数 67
text プロパティ 618
textColor プロパティ 619
TextField
 _alpha プロパティ 588
 _height プロパティ 597
 _highquality プロパティ 597
 _name プロパティ 601
 _parent プロパティ 605
 _quality プロパティ 607
 _rotation プロパティ 611
 _soundbuftime プロパティ 616
 _target プロパティ 618
 _url プロパティ 621
 _visible プロパティ 622
 _width プロパティ 622
 _x プロパティ 624
 _xmouse プロパティ 624
 _xscale プロパティ 625
 _y プロパティ 626
 _ymouse プロパティ 626
 _yscale プロパティ 627
 addListener() メソッド 586
 autoSize プロパティ 588
 background プロパティ 590
 backgroundColor プロパティ 591
 border プロパティ 591
 borderColor プロパティ 591
 bottomScroll プロパティ 592
 condenseWhite プロパティ 592
 embedFonts プロパティ 593
 getDepth() メソッド 594
 getNewTextFormat() メソッド 595
 getTextFormat() メソッド 595
 hscroll プロパティ 597
 html プロパティ 598

htmlText プロパティ 599
length プロパティ 599
maxChars プロパティ 599
maxscroll プロパティ 600
maxscroll プロパティ 600
multiline プロパティ 601
password プロパティ 606
removeListener() メソッド 607
removeTextField() メソッド 608
replaceSel() メソッド 609
replaceText() メソッド 610
scroll プロパティ 611
selectable プロパティ 612
setNewTextFormat() メソッド 613
setTextFormat() メソッド 614
tabEnabled プロパティ 616
tabIndex プロパティ 617
text プロパティ 618
textColor プロパティ 619
textHeight プロパティ 619
textWidth プロパティ 620
type プロパティ 620
variable プロパティ 621
wordWrap プロパティ 623

TextField クラス 581
onChanged イベント 602
onKillFocus イベント 602
onScroller イベント 603
onSetFocus イベント 605

TextFormat
align プロパティ 630
blockIndent プロパティ 630
bold プロパティ 631
bullet プロパティ 631
color プロパティ 632
font プロパティ 632
getTextExtent() メソッド 633
indent プロパティ 635
italic プロパティ 636
leading プロパティ 636
leftMargin プロパティ 637
rightMargin プロパティ 637
size プロパティ 638
tabStops プロパティ 638
target プロパティ 639
TextFormat() コンストラクタ 640
underline プロパティ 641
url プロパティ 642

TextFormat クラス 627
TextFormat() コンストラクタ 640

textHeight プロパティ 619
textWidth プロパティ 620
this プロパティ 89
throw ステートメント 185
toggleHighQuality 関数 69
toLowerCase() メソッド 576
toString() メソッド 233、236、314、320、360、489、
503、576、685
toUpperCase() メソッド 577
trace 関数 69
trackAsMenu プロパティ 256、462
true 定数 13
try..catch..finally ステートメント 186
type プロパティ 620
typeof 演算子 149

U

undefined 定数 14
underline プロパティ 641
unescape 関数 70
UNIQUESORT プロパティ 234
unloadClip() メソッド 484
unloadMovie 関数 71
unloadMovie() メソッド 462
unloadMovieNum 関数 71
unshift() メソッド 234
unwatch() メソッド 504
UP プロパティ 345
url プロパティ 642
useCodepage プロパティ 580
UTC() メソッド 315

V

valueOf() メソッド 237、315、489、505、578
var ステートメント 190
variable プロパティ 621
version プロパティ 281
Video
close() メソッド 644
pause() メソッド 645
play() メソッド 646
resume() メソッド 646
stop() メソッド 647
Video クラス 642
onStatus イベント 645
videoMIMETypes プロパティ 281
void 演算子 150

W

watch() メソッド 506
while ステートメント 191
width プロパティ 563
with ステートメント 192
wordWrap プロパティ 623

X

XML

addRequestHeader() メソッド 651
contentType プロパティ 652
createElement() メソッド 652
createTextNode() メソッド 653
docTypeDecl プロパティ 654
getBytesLoaded() メソッド 655
getBytesTotal() メソッド 656
ignoreWhite プロパティ 656
load() メソッド 658
loaded プロパティ 659
parseXML() メソッド 662
send() メソッド 662
sendAndLoad() メソッド 663
status プロパティ 665
XML() コンストラクタ 666
xmlDecl プロパティ 667
XML クラス 647
 onData イベント 660
 onLoad イベント 661
XML() コンストラクタ 666
xmlDecl プロパティ 667
XMLNode
 appendChild() メソッド 670
 attributes プロパティ 671
 childNodes プロパティ 672
 cloneNode() メソッド 673
 firstChild プロパティ 674
 hasChildNodes() メソッド 676
 insertBefore() メソッド 676
 lastChild プロパティ 677
 nextSibling プロパティ 678
 nodeName プロパティ 679
 nodeType プロパティ 680
 nodeValue プロパティ 681
 parentNode プロパティ 683
 previousSibling プロパティ 683
 removeNode() メソッド 684
 toString() メソッド 685
XMLNode クラス 668

XMLSocket

close() メソッド 688
connect() メソッド 688
send() メソッド 693
XMLSocket() コンストラクタ 693
XMLSocket クラス 686
 onClose イベント 690
 onConnect イベント 690
 onData イベント 691
 onXML イベント 692
XMLSocket() コンストラクタ 693

え

演算子 91

か

加算結合(ストリング)演算子 115

く

グローバル関数 15
グローバルプロパティ 72

こ

コンパイラディレクティブ 7

し

終了コマンド 207

す

ステートメント 150

て

定数 11