

# FLASH® LITE™ 1.x ACTIONSCRIPT™ リファレンスガイド

© 2007 Adobe Systems Incorporated. All rights reserved.

## Adobe® Flash® Lite™ 1.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、あくまでも参照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ありません。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트워크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트워크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트워크または画像を新しいア트워크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/amm/>) 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 ビデオ技術を使用しています。© 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Visual SourceSafe は米国または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/](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

はじめに.....	9
大部分の ActionScript エLEMENTの項目例.....	9
サンプルフォルダ.....	10
表記規則.....	10
<b>第1章：Flash Lite のグローバル関数.....</b>	<b>11</b>
call().....	13
chr().....	14
duplicateMovieClip().....	14
eval().....	16
getProperty().....	17
getTimer().....	17
getURL().....	18
gotoAndPlay().....	20
gotoAndStop().....	21
ifFrameLoaded().....	22
int().....	23
length().....	23
loadMovie().....	24
loadMovieNum().....	25
loadVariables().....	27
loadVariablesNum().....	28
mbchr().....	29
mblength().....	30
mbord().....	30
mbsubstring().....	31
nextFrame().....	32
nextScene().....	32
Number().....	33
on().....	34
ord().....	35
play().....	35
prevFrame().....	36
prevScene().....	36
random().....	37
removeMovieClip().....	38

set()	38
setProperty()	39
stop()	40
stopAllSounds()	40
String()	41
substring()	42
tellTarget()	42
toggleHighQuality()	43
trace()	44
unloadMovie()	45
unloadMovieNum()	46
<b>第2章: Flash Lite のプロパティ</b>	<b>47</b>
/( スラッシュ )	48
_alpha	49
_currentframe	50
_focusrect	50
_framesloaded	51
_height	51
_highquality	52
_level	52
maxscroll	53
_name	54
_rotation	54
scroll	55
_target	56
_totalframes	56
_visible	57
_width	57
_x	58
_xscale	58
_y	59
_yscale	60
<b>第3章: Flash Lite のステートメント</b>	<b>61</b>
break	62
case	63
continue	64
do..while	65
else	66
else if	67
for	68
if	69
switch	69
while	71

<b>第 4 章: Flash Lite の演算子</b> .....	<b>73</b>
add ( ストリング連結 ).....	76
+= ( 加算後代入 ).....	76
and.....	77
= ( 代入 ).....	78
/* ( ブロックコメント ).....	78
,( カンマ ).....	79
//( コメント ).....	80
?: ( 条件演算子 ).....	81
-- ( デクリメント ).....	82
/( 除算 ).....	82
/=( 除算後代入 ).....	83
.( ドット ).....	84
++ ( インクリメント ).....	84
&& ( 論理積 (AND)).....	86
!( 否定 (NOT)).....	86
( 論理和 (OR)).....	87
%( 剰余 ).....	88
%= ( 剰余を代入 ).....	89
*= ( 乗算後代入 ).....	89
* ( 乗算 ).....	90
+ ( 数値加算 ).....	91
==( 数値等価 ).....	91
> ( より大きい - 数値 ).....	92
>= ( より大きいか等しい - 数値 ).....	93
<> ( 数値不等価 ).....	93
< ( より小さい - 数値 ).....	94
<= ( より小さいか等しい - 数値 ).....	95
() ( 括弧 ).....	95
"" ( ストリング区切り記号 ).....	96
eq ( ストリング等価 ).....	97
gt ( より大きい - ストリング ).....	97
ge ( より大きいか等しい - ストリング ).....	98
ne ( ストリング不等価 ).....	99
lt ( より小さい - ストリング ).....	100
le ( より小さいか等しい - ストリング ).....	101
- ( 減算 ).....	102
-= ( 減算後代入 ).....	103
<b>第 5 章: Flash Lite 固有の言語エレメント</b> .....	<b>105</b>
機能.....	108
fscommand().....	116
fscommand2().....	117
<b>索引</b> .....	<b>149</b>



# はじめに

本マニュアルでは、Macromedia® Flash® Lite™ 1.0 および Macromedia® Flash® Lite™ 1.1 (まとめて Flash Lite 1.x と呼びます) 向けのアプリケーションを開発する際に使用する ActionScript エレメントのシンタックスと使用方法について説明します。Flash Lite 1.x ActionScript は、アドビシステムズ社の Macromedia® Flash® 4 で使用されたバージョンの ActionScript に基づいています。スクリプト内の例を使用するには、本マニュアルからコード例をコピーし、スクリプトペインまたは外部のスクリプトファイルにペーストしてください。本マニュアルには、すべての ActionScript エレメント (演算子、キーワード、ステートメント、コマンド、プロパティ、関数、クラス、およびメソッド) が記載されています。

## 大部分の ActionScript エレメントの項目例

次の項目例では、すべての ActionScript エレメントの記述に使われる規則について説明します。

### 項目のタイトル

各章の項目はアルファベット順に記載されています。アルファベット順では、大文字、先行アンダースコアなどを無視します。

### 使用できるバージョン

特に説明がない限り、「使用できるバージョン」の項には、該当するエレメントをサポートしている Flash Lite のバージョンが記載されています。

### シンタックス

この項では、コードで ActionScript を使用するための正しいシンタックスを示します。シンタックスの必須部分は、コード用のフォント (code) で示します。独自に指定する部分とデータ型の情報は、イタリックのコード用フォント (*italicized code font*) で示します。データ型は、コロン (:) で指定したコードとは異なります。コロンは常にデータ型情報の前に置きます。角括弧 ([ ]) は、オプションのパラメータを示します。

## パラメータ

この項では、シンタックスに含まれるパラメータについて説明します。

## 説明

この項では、エレメントの種類 ( 演算子や関数など )、エレメントが返す値を特定し、エレメントの使用法を説明します。

## 例

この項では、エレメントの使用法をサンプルコードで示します。

## 関連項目

この項では、関連する ActionScript 辞書項目を列挙します。

# サンプルフォルダ

実用的な ActionScript コードを含む完全な Flash Lite プロジェクトのサンプルについては、Flash Lite のサンプルとチュートリアルページ ([www.adobe.com/go/learn\\_ft\\_samples\\_and\\_tutorials\\_jp](http://www.adobe.com/go/learn_ft_samples_and_tutorials_jp)) を参照してください。ActionScript バージョンの .zip ファイルを探してダウンロードおよび解凍し、"Samples" フォルダを選択してサンプルファイルにアクセスします。

# 表記規則

本マニュアルでは、次の表記規則を使用しています。

- ActionScript コードは、Code font で表示されています。
- *Code font italic* (イタリック体のコードフォント) は ActionScript パラメータを示します。
- **Bold font (ボールドフォント)** はリテラルの入力を示します。
- コードの例にある二重引用符 (" ") は区切りストリングを示します。ただし、プログラマは単一引用符を使用することもできます。

この項ではシンタックスおよびアドビ システムズ社の Macromedia Flash Lite 1.1 ActionScript グローバル関数の使用方法を説明します。以下の関数があります。

関数	説明
<code>call()</code>	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()</code>	ASCII コード番号を文字に変換します。
<code>duplicateMovieClip()</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>eval()</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。
<code>getProperty()</code>	指定されたムービークリップの指定されたプロパティの値を返します。
<code>getTimer()</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()</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>gotoAndPlay()</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を開始します。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移動します。
<code>gotoAndStop()</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。シーンを指定しないと、再生ヘッドは現在のシーン内のフレームに送られます。
<code>ifFrameLoaded()</code>	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()</code>	小数点以下を切り捨てて整数値にします。
<code>length()</code>	指定された文字列または変数名の文字数を返します。
<code>loadMovie()</code>	元の SWF ファイルの再生中に、その SWF ファイルを Flash Lite にロードします。
<code>loadMovieNum()</code>	ロードした元の SWF ファイルの再生中に、その SWF ファイルを Flash Lite のいずれかのレベルにロードします。

関数	説明
<code>loadVariables()</code>	テキストファイルや、ColdFusion、CGI、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 Lite のいずれかのレベルの変数に値を設定します。この関数は、アクティブな SWF ファイル内の変数を新しい値で更新することもできます。
<code>loadVariablesNum()</code>	テキストファイルや、ColdFusion、CGI、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 Lite のいずれか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。
<code>mbchr()</code>	ASCII コード番号をマルチバイト文字に変換します。
<code>mblength()</code>	マルチバイト文字のストリング長を返します。
<code>mbord()</code>	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。
<code>mbsubstring()</code>	マルチバイト文字ストリングから、新しいマルチバイト文字ストリングを抽出します。
<code>nextFrame()</code>	再生ヘッドを次のフレームに送り、停止します。
<code>nextScene()</code>	再生ヘッドを次のシーンのフレーム 1 に送り、停止します。
<code>Number()</code>	式を数値に変換し、値を返します。
<code>on()</code>	イベントをトリガするユーザーイベントまたはキー押下を指定します。
<code>ord()</code>	文字を ASCII コード番号に変換します。
<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>	再生ヘッドを前のフレームに送り、停止します。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。
<code>prevScene()</code>	再生ヘッドを前のシーンのフレーム 1 に送り、停止します。
<code>removeMovieClip()</code>	<code>duplicateMovieClip()</code> を使用して作成したムービークリップのうち指定したものを削除します。
<code>set()</code>	変数に値を代入します。
<code>setProperty()</code>	ムービー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>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>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()</code>	ストリングの一部を抽出します。
<code>tellTarget()</code>	<i>statement(s)</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。

関数	説明
<code>toggleHighQuality()</code>	Flash Lite 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、SWF ファイルの再生は遅くなります。
<code>trace()</code>	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。
<code>unloadMovie()</code>	<code>loadMovie()</code> <code>loadMovieNum()</code> 、または <code>duplicateMovieClip()</code> を使用してロードしたムービークリップを Flash Lite から削除します。
<code>unloadMovieNum()</code>	<code>loadMovie()</code> <code>loadMovieNum()</code> 、または <code>duplicateMovieClip()</code> を使用してロードしたムービークリップを Flash Lite のレベルから削除します。

## call()

### 使用できるバージョン

Flash Lite 1.0

### シンタックス

`call(frame)`

`call(movieClipInstance: frame)`

### パラメータ

*frame* タイムラインのフレームのラベルまたは番号。

*movieClipInstance* ムービークリップのインスタンス名。

### 説明

関数。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にローカル変数は存在しません。call() 関数には 2 つの形式があります。

- デフォルトの形式では、call() 関数が実行された同じタイムラインの指定のフレームに対してスクリプトを実行します。再生ヘッドはそのフレームに移動しません。
- 指定クリップインスタンス形式では、指定されたムービークリップインスタンスのフレームで、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。



call() 関数は同期的に実行されます。call() 関数の後の ActionScript は、指定されたフレームのすべての ActionScript が完了するまでは実行されません。

## 例

次の例ではスクリプトが myScript フレームに対して実行されます。

```
// ラベル "myScript" のフレームに関数を実行します
thisFrame = "myScript";
trace ("Calling the script in frame: " + thisFrame);

// 同じタイムラインの他のフレームに関数を実行します
call("myScript");
```

# chr()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
chr(number)
```

## パラメータ

*number* ASCII コード番号。

## 説明

ストリング関数。ASCII コード番号を文字に変換します。

## 例

次の例では、番号 65 を文字 A に変換し、それを変数 myVar に割り当てます。

```
myVar = chr(65);
trace (myVar); // 出力 : A
```

# duplicateMovieClip()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
duplicateMovieClip(target, newname, depth)
```

## パラメータ

*target* 複製するムービークリップのターゲットパス。

*newname* 複製したムービークリップのインスタンス名。

*depth* 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を示します。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、より高い深度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既存のムービークリップを上書きしないように、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る必要があります。

## 説明

関数。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。戻り値はありません。複製ムービークリップの再生ヘッドは、元の ( 親の ) 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。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。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。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() 関数を使用します。新しいムービークリップは、*newname* オペランドとして渡されるストリングを使用して参照します。

## 例

次の例では、originalClip というムービークリップが複製され、newClip という新しいクリップが深度 10 に作成されます。新しいクリップの x 位置は 100 ピクセルに設定されます。

```
duplicateMovieClip("originalClip", "newClip", 10);
setProperty("newClip", _x, 100);
```

次の例では duplicateMovieClip() を for ループ内で使用していくつかの新しいムービークリップを一度に作成します。インデックス変数で、重なりが一番上の深度を保持します。複製ムービーそれぞれの名前の末尾に重なり深度に対応する数値が付きます (clip1、clip2、clip3)。

```
for (i = 1; i <= 3; i++) {
    newName = "clip" + i;
    duplicateMovieClip("originalClip", newName); }
```

## 関連項目

[removeMovieClip\(\)](#)

# eval()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
eval(expression)
```

## パラメータ

*expression* 参照すべき変数、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を示す文字列。

## 説明

関数。変数、プロパティ、オブジェクト、ムービークリップに名前アクセスします。*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。`eval()` を使用して配列をシミュレートしたり、変数の値を動的に設定および取得することができます。

## 例

次の例では、`eval()` を使用して式 "piece" + x の値を決定します。結果が変数名 `piece3` であるため、`eval()` は変数の値を返し、それを `y` に割り当てます。

```
piece3 = "dangerous";  
x = 3;  
y = eval("piece" + x);  
trace(y);// 出力 : dangerous
```

次の例は、配列をシミュレートする方法を示しています。

```
name1 = "mike";  
name2 = "debbie";  
name3 = "logan";  
for(i = 1; i <= 3; i++) {  
    trace (eval("name" + i));// 出力 : mike, debbie, logan  
}
```

# getProperty()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
getProperty(my_mc, property)
```

## パラメータ

*my\_mc* 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* ムービークリップのプロパティ。

## 説明

関数。ムービークリップ *my\_mc* の指定したプロパティの値を返します。

## 例

次の例では、ルートムービータイムラインの *my\_mc* ムービークリップの水平軸座標 (*\_x*) を取得します。

```
xPos = getProperty("person_mc", _x);  
trace (xPos); // 出力 : -75
```

## 関連項目

[setProperty\(\)](#)

# getTimer()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
getTimer()
```

## パラメータ

なし

## 説明

関数。SWF ファイルの再生を開始してからの経過時間をミリ秒単位で返します。

## 例

次の例では、*timeElapsed* 変数に SWF ファイルの再生開始から経過した時間 (ミリ秒) を設定します。

```
timeElapsed = getTimer();  
trace (timeElapsed); // 出力 : ムービーが再生されている時間 (ミリ秒)
```

# getURL()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
getURL(url [ , window [ , "variables" ] ])
```

## パラメータ

*url* ドキュメントを取得するための URL。

*window* 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オプションのパラメータ。



携帯電話のブラウザでは複数のウィンドウがサポートされないため、*window* パラメータは Flash Lite アプリケーションには指定できません。

空の文字列を入力するか、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` - 現在のウィンドウ内の現在のフレームを指定します。
- `_blank` - 新規ウィンドウを指定します。
- `_parent` - 現在のフレームの親を指定します。
- `_top` - 現在のウィンドウ内の最上位のフレームを指定します。

*variables* 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 ヘッダー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。

## 説明

関数。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 にある別のアプリケーションに変数を渡します。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。

Flash Lite 1.0 が認識できるのは、HTTP、HTTPS、mailto、および tel プロトコルだけです。Flash Lite 1.1 はこれらのプロトコルに加えて、ファイル、SMS (ショートメッセージサービス)、MMS (マルチメディアメッセージサービス) プロトコルを認識できます。

Flash Lite は呼び出しをオペレーティングシステムに渡し、オペレーティングシステムは指定されたプロトコル向けに登録されたデフォルトのアプリケーションでその呼び出しを処理します。

1つのフレームまたは1つのイベントハンドラに対して、1つの `getURL()` 関数が処理されます。

一部のハンドセットでは `getURL()` 関数がキーイベントのみに制限されます。この場合、`getURL()` コールはキーイベントハンドラでトリガされた場合にのみ処理されます。このような場合でも、1つのイベントハンドラに対して1つの `getURL()` 関数のみが処理されます。

## 例

次の `ActionScript` では、`Flash Lite Player` によってデフォルトのブラウザ上に `mobile.example.com` を開きます。

```
myURL = "http://mobile.example.com";
on(keyPress "1") {
    getURL(myURL);
}
```

`GET` または `POST` を使用して、現在のタイムラインから変数を送信することもできます。次の例では、`GET` メソッドを使用して、変数を `URL` に追加します。

```
firstName = "Gus";
lastName = "Richardson";
age = 92;
getURL("http://www.example.com", "_blank", "GET");
```

次の `ActionScript` では `POST` を使用して `HTTP` ヘッダー内で変数を送信します。

```
firstName = "Gus";
lastName = "Richardson";
age = 92;
getURL("http://www.example.com", "POST");
```

ボタンの関数を割り当てて、`address`、`subject`、`body` テキストフィールドに既に入力された状態で電子メール作成ウィンドウを開くことができます。ボタンの関数を割り当てる場合、`Shift-JIS` または英語文字エンコーディングの場合は方法 1、英語文字のみのエンコーディングの場合は方法 2 を使用します。

方法 1: 次の例のように、各パラメータの変数を設定します。

```
on (release, keyPress "#"){
    subject = "email subject";
    body = "email body";
    getURL("mailto:somebody@anywhere.com", "", "GET");
}
```

方法 2: 次の例に従って、`getURL()` 関数内の各パラメータを定義します。

```
on (release, keyPress "#"){
    getURL("mailto:somebody@anywhere.com?cc=cc@anywhere.com&bcc=bcc@anywhere.com&subject=I am the email subject&body=I am the email body");
}
```

方法 1 では `URL` が自動的にエンコーディングされ、方法 2 では `URL` のスペースが保持されます。たとえば、方法 1 を使用した場合、結果の `URL` は次のようになります。

```
email+subject  
email+body
```

一方、方法 2 では結果のストリングは次のようになります。

```
email subject  
email body
```

次の例では tel プロトコルが使用されています。

```
on (release, keyPress "#"){  
    getURL("tel:117");  
}
```

次の例では、getURL() を使用して、SMS メッセージを送信します。

```
mySMS = "sms:4156095555?body=sample sms message";  
on(keyPress "5") {  
    getURL(mySMS);  
}
```

次の例では、getURL() を使用して、MMS メッセージを送信します。

```
// MMS の例  
myMMS = "mms:4156095555?body=sample mms message";  
on(keyPress "6") {  
    getURL(myMMS);  
}
```

次の例では、getURL() を使用してローカルファイルシステムに格納されたテキストファイルを開きます。

```
// file プロトコルの例  
filePath = "file:///c:/documents/flash/myApp/myvariables.txt";  
on(keyPress "7") {  
    getURL(filePath);  
}
```

## gotoAndPlay()

### 使用できるバージョン

Flash Lite 1.0

### シンタックス

```
gotoAndPlay([scene,] frame)
```

### パラメータ

*scene* 再生ヘッドを送る先のシーンの名前を示すストリング。このパラメータはオプションです。

*frame* 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フレームのラベルを表すストリング。

## 説明

関数。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移動します。

*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そ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。

## 例

次の例では、`gotoAndPlay()` が割り当てられたボタンをユーザーがクリックすると、再生ヘッドが現在のシーンのフレーム 16 に移動し、SWF ファイルの再生が開始します。

```
on(keyPress "7") {  
    gotoAndPlay(16);  
}
```

# gotoAndStop()

## 使用できるバージョン

Flash 1.0

## シンタックス

```
gotoAndStop([scene,] frame)
```

## パラメータ

*scene* 再生ヘッドを送る先のシーンの名前を示す文字列。このパラメータはオプションです。

*frame* 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フレームのラベルを表す文字列。

## 説明

関数。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。シーンを指定しないと、再生ヘッド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そ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。

## 例

次の例では、`gotoAndStop()` が割り当てられたボタンをユーザーがクリックすると、再生ヘッドが現在のシーンのフレーム 5 に移動し、SWF ファイルの再生が停止します。

```
on(keyPress "8") {  
    gotoAndStop(5);  
}
```

# ifFrameLoaded()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

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

## パラメータ

*scene* ロードするシーンの名前を指定するオプションのストリング。

*frame* 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の番号またはフレームのラベル。

*statement(s)* 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。

## 説明

関数。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 ファイルの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には、ifFrameLoaded 関数を使用します。[\\_framesloaded](#) プロパティを使用して、外部 SWF ファイルのダウンロードの進行状況をチェックすることもできます。[\\_framesloaded](#) を使用する場合と ifFrameLoaded を使用する場合の違いは、[\\_framesloaded](#) の場合は独自の if または else ステートメントを追加できる点です。

## 例

次の例では、ifFrameLoaded 関数を使用して SWF ファイルのフレーム 10 がロードされたかチェックします。フレームがロードされると、trace() コマンドによって [出力] パネルに "frame number 10 is loaded" と出力されます。変数 output は、frame loaded: 10 という値に設定されます。

```
ifFrameLoaded(10) {  
    trace ("frame number 10 is loaded");  
    output = "frame loaded: 10";  
}
```

## 関連項目

[\\_framesloaded](#)

# int()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
int(value)
```

## パラメータ

*value* 切り捨てて整数にする数値またはストリング。

## 説明

関数。小数点以下を切り捨てて整数値にします。

## 例

次の例では、`distance` と `myDistance` 変数内の数値を切り捨てます。

```
distance = 6.04 - 3.96;  
//trace ("distance = " add distance add " and rounded to:" add int(distance));  
// 出力 : distance = 2.08 and rounded to: 2  
myDistance = "3.8";  
//trace ("myDistance = " add int(myDistance));  
// 出力 : 3
```

# length()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
length(expression)
```

```
length(variable)
```

## パラメータ

*expression* ストリング。

*variable* 変数の名前。

## 説明

ストリング関数。指定されたストリングまたは名前の変数の文字数を返します。

## 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。

次の例では、電子メールアドレスが 6 文字以上であることを検査します。

```
email = "someone@example.com";  
if (length(email) > 6) {  
    //trace ("email appears to have enough characters to be valid");  
}
```

# loadMovie()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
loadMovie(url, target [, method])
```

## パラメータ

*url* ロードする SWF ファイルの絶対 URL または相対 URL を指定するストリング。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* 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 ファイルに置き換えられます。

*method* 変数を送信するための HTTP メソッドを指定するオプションのストリングパラメータ。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、独立した HTTP ヘッダー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。

## 説明

関数。元の SWF ファイルの再生中に、SWF ファイルを Flash Lite にロードします。

SWF ファイルを指定したレベルにロードするには、`loadMovie()` の代わりに `loadMovieNum()` 関数を使用します。

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合わせされます。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合わせされます。

`loadMovie()` を使ってロードしたムービーを削除するには、`unloadMovie()` 関数を使用します。

## 例

次の例では、ステージにある `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");  
// 同等のステートメント : loadMovie("circle.swf", _level0.mySquare);
```

## 関連項目

[\\_level](#)、[loadMovieNum\(\)](#)、[unloadMovie\(\)](#)、[unloadMovieNum\(\)](#)

# loadMovieNum()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
loadMovieNum(url, level [, method])
```

## パラメータ

*url* ロードする SWF ファイルの絶対 URL または相対 URL を指定する文字列。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 Lite 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使用する際には、すべての SWF ファイルを同じフォルダに置く必要があります。ファイル名にフォルダやドライブの指定を含めることはできません。

*level* SWF ファイルのロード先の Flash Lite のレベルを指定する整数。

*method* 変数を送信するための HTTP メソッドを指定するオプションの文字列パラメータ。値は GET または POST です。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いときに使用します。POST メソッドは、別の HTTP ヘッダー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。

## 説明

関数。ロードした元の SWF ファイルの再生中に、その SWF ファイルを Flash Lite のいずれかのレベルにロードします。

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() 関数を使用します。

Flash Lite 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 Lite 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 `_levelN` 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に、任意のレベル番号を指定できます。既に SWF ファイルがロードされているレベルに SWF ファイルをロードし、新しい SWF ファイルで既存のファイルを置き換えることができます。SWF ファイルをレベル 0 にロードすると、Flash Lite 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() 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。

## 例

次の例では、SWF ファイルをレベル 2 にロードします。

```
loadMovieNum("http://www.someserver.com/flash/circle.swf", 2);
```

## 関連項目

[\\_level](#)、[loadMovie\(\)](#)、[unloadMovieNum\(\)](#)

# loadVariables()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
loadVariables(url, target [, variables])
```

## パラメータ

*url* 変数が存在する絶対 URL または相対 URL を示す文字列。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* 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。

*variables* 変数を送るための HTTP メソッドを指定するオプションの文字列パラメータ。パラメータは文字列 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 ヘッダー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。

## 説明

関数。テキストファイルや、ColdFusion、CGI、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 사용되는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=Adobe&address=600+Townsend&city=San+Francisco&zip=94103
```

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

## 例

次の例では、テキストファイルとサーバーから変数をロードします。

```
// ローカルファイルシステム (Symbian Series 60) 上のテキストファイルから変数をロードします
on(release, keyPress "1") {
    filePath = "file:///c:/documents/flash/myApp/myvariables.txt";
    loadVariables(filePath, _root);
}
```

```
// 変数を (サーバーから) ムービークリップにロードします
urlPath = "http://www.someserver.com/myvariables.txt";
loadVariables(urlPath, "myClip_mc");
```

## 関連項目

[loadMovieNum\(\)](#)、[loadVariablesNum\(\)](#)、[unloadMovie\(\)](#)

# loadVariablesNum()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
loadVariablesNum(url, level [, variables])
```

## パラメータ

*url* ロードする変数が存在する絶対 URL または相対 URL を示す文字列。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* 変数を受け取る Flash Lite のレベルを指定する整数。

*variables* 変数を送るための HTTP メソッドを指定するオプションの文字列パラメータ。パラメータは文字列 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 ヘッダー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。

## 説明

関数。テキストファイルや、ColdFusion、CGI、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 Lite のいずれか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。次の例では、複数の変数を定義します。

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

通常、Flash Lite は1つの SWF ファイルを表示した後に閉じます。loadVariablesNum() 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。

ターゲットのムービークリップに変数をロードするには、loadVariablesNum() 関数の代わりに loadVariables() 関数を使用します。

## 関連項目

[getURL\(\)](#)、[loadMovie\(\)](#)、[loadMovieNum\(\)](#)、[loadVariables\(\)](#)

# mbchr()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
mbchr(number)
```

## パラメータ

*number* マルチバイト文字に変換する数値。

## 説明

ストリング関数。ASCII コード番号をマルチバイト文字に変換します。

## 例

次の例では、ASCII コード番号を対応するマルチバイト文字に変換します。

```
trace (mbchr(65)); // 出力 : A  
trace (mbchr(97)); // 出力 : a  
trace (mbchr(36)); // 出力 : $
```

```
myString = mbchr(51) - mbchr(49);  
trace ("result = " add myString); // 出力 : result = 2
```

## 関連項目

[mblength\(\)](#)、[mbsubstring\(\)](#)

# mblength()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`mblength(string)`

## パラメータ

*string* ストリング。

## 説明

ストリング関数。マルチバイト文字のストリング長を返します。

## 例

次の例では、`myString` 変数内のストリングの長さを表示します。

```
myString = mbchr(36) add mbchr(50);  
trace ("string length = " add mblength(myString));  
// 出力 : string length = 2
```

## 関連項目

[mbchr\(\)](#)、[mbsubstring\(\)](#)

# mbord()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`mbord(character)`

## パラメータ

*character* マルチバイト文字コード番号に変換する文字

## 説明

ストリング関数。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。

## 例

次の例では、myString 変数内の文字をマルチバイト文字列に変換します。

```
myString = "A";
trace ("ord = " add mbord(myString));// 出力 :

myString = "$120";
for (i=1; i<=length(myString); i++)
    char = substring(myString, i, 1);
    trace ("char ord = " add mbord(char));// 出力 : 36, 49, 50, 48
}
```

## 関連項目

[mbchr\(\)](#)、[mbsubstring\(\)](#)

# mbsubstring()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

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

## パラメータ

*value* 文字を抽出するマルチバイト文字の文字列。

*index* 抽出する最初の文字の番号。

*count* 抽出する文字列に含まれる文字数。インデックス文字を除きます。

## 説明

文字列関数。マルチバイト文字の文字列から、マルチバイト文字の文字列を抽出します。

## 例

次の例では、myString 変数に含まれる文字列から新しいマルチバイト文字列を抽出します。

```
myString = mbchr(36) add mbchr(49) add mbchr(50) add mbchr(48);
trace (mbsubstring(myString, 0, 2));// 出力 : $1
```

## 関連項目

[mbchr\(\)](#)

# nextFrame()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
nextFrame()
```

## パラメータ

なし

## 説明

関数。再生ヘッドを次のフレームに送り、停止します。

## 例

次の例では、ユーザーがボタンをクリックすると、再生ヘッドが次のフレームに移動して停止します。

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

## 関連項目

[prevFrame\(\)](#)

# nextScene()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
nextScene()
```

## パラメータ

なし

## 説明

関数。再生ヘッドを次のシーンのフレーム 1 に送り、停止します。

## 例

次の例では、ユーザーがボタンを離すと、再生ヘッドが次のシーンのフレーム 1 に移動します。

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

## 関連項目

[prevScene\(\)](#)

# Number()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`Number(expression)`

## パラメータ

*expression* 数値に変換される式。

## 説明

関数。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* が数値であれば、戻り値は *expression* です。
- *expression* がブール値である場合、戻り値は *expression* が `true` であれば `1` であり、*expression* が `false` であれば `0` です。
- *expression* が文字列である場合、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とする場合があります。
- *expression* が `undefined` であれば、戻り値は `-1` です。

## 例

次の例では、`myString` 変数内の文字列を数値に変換して、その数値を `myNumber` 変数に格納し、数値に `5` を加算して、結果を変数 `myResult` に格納します。最後の行はブール値の場合に `Number()` を呼び出したときの結果を示します。

```
myString = "55";  
myNumber = Number(myString);  
myResult = myNumber + 5;  
  
trace (myResult); // 出力 :  
  
trace (Number(true)); // 出力 :
```

# on()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
on(event) {  
    // statement(s)  
}
```

## パラメータ

*statement(s)* イベントが発生したときに実行する指示。

*event* このトリガはイベントと呼ばれます。ユーザー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、*event* 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ボタンを押した場合。
- `release` ポインタがボタン上にあるときにボタンを離した場合。
- `rollOut` ポインタがボタン領域の外側に移動した場合。
- `rollOver` ポインタがボタン上に移動した場合。
- `keyPress ("key")` 指定されたキーを押した場合。パラメータのうち `key` の部分には、キーコードまたはキー定数を指定します。

## 説明

イベントハンドラ。関数をトリガするユーザーイベントまたはキー押下を指定します。サポートされないイベントもあります。

## 例

次のコードでは、ユーザーが 8 キーを押すと `myText` フィールドが下に 1 行スクロールされます。スクロールする前に `maxscroll` に対してテストされます。

```
on (keyPress "8") {  
    if (myText.scroll < myText.maxscroll) {  
        myText.scroll++;  
    }  
}
```

# ord()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`ord(character)`

## パラメータ

*character* ASCII コード番号に変換する文字。

## 説明

ストリング関数。文字を ASCII コード番号に変換します。

## 例

次の例では、`ord()` 関数を使用して文字 **A** の ASCII コードを表示します。

```
trace ("multibyte number = " + add ord("A")); // 出力 : multibyte number = 65
```

# play()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`play()`

## パラメータ

なし

## 説明

関数。タイムライン内で再生ヘッドを先へ進めます。

## 例

次の例では、`if` ステートメントを使用して、ユーザーが入力した名前の値をチェックします。ユーザーが **Steve** と入力した場合、`play()` 関数が呼び出されて再生ヘッドがタイムライン内で前に進みます。ユーザーが **Steve** 以外を入力した場合は、**SWF** ファイルは再生されず、`alert` という変数名のテキストフィールドが表示されます。

```
stop();  
if (name == "Steve") {  
    play();  
} else {  
    alert="You are not Steve!";  
}
```

# prevFrame()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
prevFrame()
```

## パラメータ

なし

## 説明

関数。再生ヘッドを直前のフレームに戻し、停止します。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

以下のハンドラが割り当てられたボタンをユーザーがクリックすると、再生ヘッドが前のフレームに移動します。

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

## 関連項目

[nextFrame\(\)](#)

# prevScene()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
prevScene()
```

## パラメータ

なし

## 説明

関数。再生ヘッドを直前のシーンのフレーム1に送り、停止します。

## 例

次の例では、以下のハンドラが割り当てられたボタンをユーザーがクリックすると、再生ヘッドが前のシーンに移動します。

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

## 関連項目

[nextScene\(\)](#)

# random()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
random(value)
```

## パラメータ

*value* 整数。

## 説明

関数; 0 から *value* パラメータで指定された整数未満までのランダムな整数を返します。

## 例

次の例では、範囲を指定する整数に基づいて数値を生成します。

```
// 0 ~ 4 のランダムな数値を選びます
myNumber = random(5);
trace (myNumber); // 出力 : 0、1、2、3、または 4
```

```
// 5 ~ 9 のランダムな数値を選びます
myNumber = random(5) + 5;
trace (myNumber); // 出力 : 5、6、7、8、または 9
```

次の例では、数値を生成し、変数名として評価されるストリングの末尾にその数値を連結します。この例は Flash Lite 1.1 のシンタックスを使用して配列をシミュレートする方法を示しています。

```
// リストからランダムな名前を選択します
myNames1 = "Mike";
myNames2 = "Debbie";
myNames3 = "Logan";

ran = random(3) + 1;
ranName = "myNames" + ran;
trace (eval(ranName)); // 出力 : mike、debbie、または logan
```

# removeMovieClip()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
removeMovieClip(target)
```

## パラメータ

*target* `duplicateMovieClip()` で作成したムービークリップのインスタンスのターゲットパス。

## 説明

関数。`duplicateMovieClip()` を使用して作成したムービークリップのうち指定したものを削除します。

## 例

次の例では、`second_mc` という名前の複製ムービークリップを削除します。

```
duplicateMovieClip("person_mc", "second_mc", 1);
second_mc._x = 55;
second_mc._y = 85;
removeMovieClip("second_mc");
```

# set()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
set(variable, expression)
```

## パラメータ

*variable* *expression* パラメータの値を格納する識別子。

*expression* 変数に割り当てられる値。

## 説明

ステートメント。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ことで、ユーザーが実行した操作情報を記録して保存できます。また、SWF ファイルの再生中に変化した値を記録したり、条件が `true` なのか `false` なのかを評価することもできます。

変数は、文字列、数値、ブール値、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。

## 例

次の例では、`orig_x_pos` という変数を設定します。この変数には `ship` ムービークリップの元の `x` 座標が格納され、後で SWF ファイルの船の位置を開始位置にリセットします。

```
on(release) {
    set("orig_x_pos", getProperty("ship", _x));
}
```

前のコードの結果は次のコードと同様です。

```
on(release) {
    orig_x_pos = ship._x;
}
```

# setProperty()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`setProperty(target, property, value/expression)`

## パラメータ

*target* 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* 設定するプロパティ。

*value* プロパティの新しいリテラル値。

*expression* プロパティの新しい値として評価される式。

## 説明

関数。ムービー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。

## 例

次のステートメントでは、ユーザーがこのイベントハンドラと関連付けられたボタンをクリックしたときに、`star` ムービークリップの `_alpha` プロパティを 30 パーセントに設定します。

```
on(release) {
    setProperty("star", _alpha, "30");
}
```

## 関連項目

[getProperty\(\)](#)

# stop()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
stop()
```

## パラメータ

なし

## 説明

関数。再生中の SWF ファイルを停止します。この関数は、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。

## 例

次のステートメントでは、ユーザーがこのイベントハンドラと関連付けられたボタンをクリックしたときに stop() 関数を呼び出します。

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

# stopAllSounds()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
stopAllSounds()
```

## パラメータ

なし

## 説明

関数。再生ヘッドを停止せずに、SWF ファイルで再生中のすべてのサウンドを停止します。ストリーミングするために設定されたサウンドは、そのサウンドが含まれ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

次のコードは、クリックしたときに SWF ファイルのすべてのサウンドをオフにするボタンに適用できます。

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

# String()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`String(expression)`

## パラメータ

*expression* スtringに変換される式。

## 説明

関数。指定されたパラメータのString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Stringは数値のテキスト表現です。
- *expression* がStringである場合、返されるStringは *expression* です。
- *expression* がブール値の場合、返されるString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。

## 例

次の例では、`birthYearNum` を 1976 に設定し、それを `String()` 関数を使用してStringに変換し、`eq` 演算子を使用してString "1976" と比較します。

```
birthYearNum = 1976;  
birthYearStr = String(birthYearNum);  
if (birthYearStr eq "1976") {  
    trace ("birthYears match");  
}
```

# substring()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
substring(string, index, count)
```

## パラメータ

*string* 新しいストリングを抽出する元のストリング。

*index* 抽出する最初の文字の番号。

*count* 抽出するストリングに含める文字数。インデックス文字を除きます。

## 説明

関数。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 クラスのメソッドは 0 から始まります。

## 例

次の例では、ストリング "Hello World" から最初の 5 文字を抽出します。

```
origString = "Hello World!";  
newString = substring(origString, 0, 5);  
trace (newString);// 出力 : Hello
```

# tellTarget()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

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

## パラメータ

*target* 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)* 指定したタイムラインをターゲットに実行される指示。

## 説明

関数。 *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` 関数は、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、`tellTarget()` をボタンに指定して、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するように指示することができます。

## 例

次の例では、`tellTarget()` でメインタイムラインの `ball` ムービークリップインスタンスを制御します。 `ball` インスタンスのフレーム1は空白であり、`stop()` 関数があるため、ステージ上では見えません。ユーザーが5キーを押すと、`tellTarget()` が `ball` の再生ヘッドに対して、アニメーションの開始位置であるフレーム2へ移動するように指示します。

```
on(keyPress "5") {
    tellTarget("ball") {
        gotoAndPlay(2);
    }
}
```

# toggleHighQuality()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
toggleHighQuality()
```

## パラメータ

なし

## 説明

関数。Flash Lite でのアンチエイリアス処理のオン / 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、SWF ファイルの再生は遅くなります。この関数は、Flash Lite 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 / 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();
}
```

# trace()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
trace(expression)
```

## パラメータ

*expression* 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 オール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。

## 説明

関数。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。

この関数は、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] パネルに表示するには、*expression* パラメータを使用します。trace() 関数は、JavaScript の alert 関数に似ています。パブリッシュ設定で [Trace アクションを省略] コマンドを使用して、書き出す SWF ファイルから trace() 関数を削除することができます。

## 例

次の例では、trace() 関数を使用して while ループの動作を確認します。

```
i = 0;
while (i++ < 5){
    trace("this is execution " + i);
}
```

# unloadMovie()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
unloadMovie(target)
```

## パラメータ

*target* ムービークリップのターゲットパス。

## 説明

関数。[loadMovie\(\)](#)、[loadMovieNum\(\)](#)、または「[duplicateMovieClip\(\)](#)」を使用してロードしたムービークリップを Flash Lite から削除します。

## 例

ユーザーが 3 のキーを押すと、次のコードが応答して `draggable_mc` ムービークリップをメインタイムラインからアンロードし、`movie.swf` をドキュメントの重なりのレベル 4 にロードします。

```
on (keypress "3") {  
    unloadMovie ("/draggable_mc");  
    loadMovieNum("movie.swf", 4);  
}
```

以下の例では、ユーザーが 3 のキーを押すとレベル 4 にロードしたムービーをアンロードします。

```
on (keypress "3") {  
    unloadMovieNum(4);  
}
```

## 関連項目

[loadMovie\(\)](#)

# unloadMovieNum()

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
unloadMovieNum(level)
```

## パラメータ

*level* ロードされたムービーのレベル (*\_levelN*)。

## 説明

関数。loadMovie()、loadMovieNum()、または「duplicateMovieClip()」を使用してロードしたムービークリップを Flash Lite から削除します。

通常、Flash Player は 1 つの SWF ファイルを表示した後に閉じます。unloadMovieNum() 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。

## 関連項目

[loadMovieNum\(\)](#)

この項では、アドビ システムズ社の Macromedia Flash Lite 1.x が認識するプロパティについて説明します。各項目は、先行アンダースコアを無視してアルファベット順に示します。次の表はプロパティの一覧です。

プロパティ	説明
/ (スラッシュ)	メインムービータイムラインへの参照を指定するか返します。
<code>_alpha</code>	ムービークリップのアルファ透明度の値を返します。
<code>_currentframe</code>	再生ヘッドが位置するタイムラインのフレームの番号を返します。
<code>_focusrect</code>	現在フォーカスがあるボタンまたはテキストフィールド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>_framesloaded</code>	動的にロードされた SWF ファイルからロードされたフレームの数を返します。
<code>_height</code>	ムービークリップの高さをピクセル単位で指定します。
<code>_highquality</code>	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>_levelN</code> のルートタイムラインへの参照を返します。 <code>loadMovieNum()</code> 関数を使用して SWF ファイルを Flash Lite Player にロードしてから、 <code>_level</code> プロパティを使用してターゲットを指定します。 <code>_levelN</code> を使用すると、ロードされている SWF ファイルを <i>N</i> によって割り当てられたレベルターゲットにすることもできます。
<code>maxscroll</code>	スクロール可能な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>_name</code>	ムービークリップのインスタンス名を返します。これはムービークリップに対してのみ適用され、メインタイムラインには使用できません。
<code>_rotation</code>	ムービークリップの元の位置からの回転角を度単位で返します。
<code>scroll</code>	変数に関連付けられたテキストフィールドでの情報の表示を制御します。 <code>scroll</code> プロパティは、テキストフィールドが内容の表示を開始する場所を定義します。その内容を設定した後、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つど、内容を更新します。
<code>_target</code>	ムービークリップインスタンスのターゲットパスを返します。

プロパティ	説明
<code>_totalframes</code>	ムービークリップ内のフレームの総数を返します。
<code>_visible</code>	ムービークリップが表示されるかどうかを示します。
<code>_width</code>	ムービークリップの幅をピクセル単位で返します。
<code>_x</code>	ムービークリップの x 座標を設定する整数を格納します。
<code>_xscale</code>	ムービークリップの基準点から適用されるように、ムービークリップの水平スケール (%) を設定します。
<code>_y</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 座標を設定する整数を格納します。
<code>_yscale</code>	ムービークリップの基準点から適用されるように、ムービークリップの垂直スケール (%) を設定します。

## /(スラッシュ)

### 使用できるバージョン

Flash Lite 1.0

### シンタックス

/

*/targetPath*

*/:varName*

### 説明

識別子。メインムービータイムラインへの参照を指定するか返します。このプロパティの機能は、Flash 5 の `_root` プロパティに似ています。

### 例

タイムラインで変数を指定するには、スラッシュシンタックスと (/) コロン (:) を組み合わせて使用します。

例 1: メインタイムラインの `car` 変数。

```
/:car
```

例 2: メインタイムラインにあるムービークリップインスタンス `mc1` の `car` 変数。

```
/mc1/:car
```

例 3: メインタイムラインにあるムービークリップインスタンス mc1 にネストされたムービークリップインスタンス mc2 の car 変数。

```
/mc1/mc2/:car
```

例 4: 現在のタイムラインにあるムービークリップインスタンス mc2 の car 変数。

```
mc2/:car
```

## **\_alpha**

### **使用できるバージョン**

Flash Lite 1.0

### **シンタックス**

*my\_mc*:\_alpha

プロパティ。 *my\_mc* 変数で指定されたムービークリップ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、デフォルト値は 100 です。\_alpha が 0 に設定されたムービークリップのオブジェクトは、非表示であってもアクティブです。たとえば、\_alpha プロパティが 0 に設定されたムービークリップ中のボタンをクリックできます。

### **例**

ボタンイベントハンドラの次のコードは、ユーザーがボタンをクリックしたときに、my\_mc ムービークリップの \_alpha プロパティを 30% に設定します。

```
on(release) {  
    tellTarget("my_mc") {  
        _alpha = 30;  
    }  
}
```

# \_currentframe

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
my_mc: _currentframe
```

## 説明

プロパティ (読み取り専用)。変数 *my\_mc* で指定したタイムラインにおいて、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 プロパティと gotoAndStop() 関数を使用して my\_mc ムービークリップの再生ヘッドを現在の位置から 5 フレーム前へ進めます。

```
tellTarget("my_mc") {  
    gotoAndStop(_currentframe + 5);  
}
```

関連項目

[gotoAndStop\(\)](#)

# \_focusrect

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
_focusrect = Boolean;
```

## 説明

プロパティ (グローバル)。現在フォーカスがあるボタンまたはテキストフィールドの回りに黄色の長方形を表示するかどうかを指定します。デフォルト値の true の場合は、ユーザーが電話またはモバイルデバイスの上矢印または下矢印キーを押して SWF ファイルのオブジェクト間をナビゲートする際に、現在フォーカスがあるボタンまたはテキストフィールドの周囲に黄色の矩形を表示します。黄色の矩形を表示しない場合は、false を指定します。

## 例

次の例は、アプリケーションでの黄色のフォーカス矩形の表示を無効にします。

```
_focusrect = false;
```

# **\_framesloaded**

## **使用できるバージョン**

Flash Lite 1.0

## **シンタックス**

*my\_mc*:\_framesloaded

## **説明**

プロパティ (読み取り専用)。動的にロードされた SWF ファイルからロードされたフレームの数です。このプロパティは、特定のフレーム以前のすべてのフレームの内容がロードされ、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断する場合に使用できます。また、大きな SWF ファイルのダウンロード中のモニタとしても便利です。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 loader の \_xscale プロパティにより、ロードの進行状況を示すプログレスバーが作成されます。

```
if (_framesloaded >= _totalframes) {  
    gotoAndPlay ("Scene 1", "start");  
} else {  
    tellTarget("loader") {  
        _xscale = (_framesloaded/_totalframes)*100;  
    }  
}
```

# **\_height**

## **使用できるバージョン**

Flash Lite 1.0

## **シンタックス**

*my\_mc*:\_height

## **説明**

プロパティ (読み取り専用)。ピクセル単位で示したムービークリップの高さです。

## 例

次のイベントハンドラのコード例は、ユーザーがマウスボタンをクリックすると、ムービークリップの高さを設定します。

```
on(release) {  
    tellTarget("my_mc") {  
        _height = 200;  
    }  
}
```

# \_highquality

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`_highquality`

## 説明

プロパティ (グローバル)。現在の SWF ファイルに適用されるアンチエイリアス処理のレベルを指定します。2 を指定すると、最高品質のアンチエイリアス処理が行われます。1 を指定すると、高品質のアンチエイリアス処理が行われます。0 を指定すると、アンチエイリアス処理は行われません。

## 例

次のステートメントは、現在の SWF ファイルに高品質のアンチエイリアス処理を適用します。

```
_highquality = 1;
```

## 関連項目

[toggleHighQuality\(\)](#)

# \_level

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`_levelN`

## 説明

識別子 `_levelN` のルートタイムラインへの参照。 `_level` プロパティを使用してターゲットを指定するには、その前に `loadMovieNum()` 関数を使用して SWF ファイルを Flash Lite Player にロードする必要があります。 `_levelN` を使用すると、 `N` で指定されたレベルにロードした SWF ファイルをターゲットにすることができます。

Flash Lite 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 `_level0` 内にロードされます。 `_level0` の SWF ファイルによって、その後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 `_level0` 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 Lite Player 内にロードする SWF ファイルごとに、 `loadMovieNum()` 関数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 ( `_level0` を含む ) を割り当てると、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、SWF ファイルをレベル 1 にロードして、ロードした SWF ファイルの再生ヘッドをフレーム 6 に停止させます。

```
loadMovieNum("mySWF.swf", 1);

// 1 フレーム以上後ろ
tellTarget(_level1) {
    gotoAndStop(6);
}
```

## 関連項目

[loadMovie\(\)](#)

# maxscroll

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
variable_name:maxscroll
```

## 説明

プロパティ (読み取り専用)。スクロール可能なテキストフィールド内の最終行が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。 `maxscroll` プロパティは、 `scroll` プロパティと共にテキストフィールド内の情報の表示を制御します。このプロパティは取得できますが、変更できません。

## 例

次のコードでは、ユーザーが **8** のキーを押すと myText テキストフィールドが下に 1 行スクロールされます。スクロールする前に maxscroll に対してテストされます。

```
on(keyPress "8") {
    if (myText:scroll < myText:maxscroll) {
        myText:scroll++;
    }
}
```

## 関連項目

[scroll](#)

# \_name

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*my\_mc*:\_name

## 説明

プロパティ。 *my\_mc* に指定したムービークリップのインスタンス名です。これはムービークリップに対してのみ適用され、メインタイムラインには使用できません。

## 例

次の例では、 bigRose ムービークリップの名前をストリングとして [ 出力 ] パネルに表示します。

```
trace(bigRose:_name);
```

# \_rotation

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*my\_mc*:\_rotation

## 説明

プロパティ。ムービークリップ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。この範囲を外れる値は、360を加算されるか360を減算され、範囲内に収まる値になるよう調整されます。たとえば、ステートメント `my_mc:_rotation = 450` は `my_mc:_rotation = 90` と同じです。

## 例

次の例では、ユーザーが2のキーを押すと、my\_mc ムービークリップが時計回りに15度回転します。

```
on (keyPress "2") {  
    my_mc:_rotation += 15;  
}
```

# scroll

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
textFieldVariableName:scroll
```

## 説明

プロパティ。変数に関連するテキストフィールドで情報の表示を制御します。scroll プロパティは、テキストフィールドが内容の表示を開始する場所を定義します。その内容を設定した後、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つど、内容を更新します。scroll プロパティを使用して、スクロールするテキストフィールドを作成したり、長い文節の特定の段落にユーザーを誘導したりできます。

## 例

次のコードは、ユーザーが2のキーを押すたびに myText テキストフィールドを上1行スクロールします。

```
on(keyPress "2") {  
    if (myText:scroll > 1) {  
        myText:scroll--;  
    }  
}
```

## 関連項目

[maxscroll](#)

# **\_target**

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*my\_mc*:\_target

## 説明

プロパティ (読み取り専用)。 *my\_mc* に指定したムービークリップインスタンスのターゲットパスを返します。

# **\_totalframes**

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*my\_mc*:\_totalframes

## 説明

プロパティ (読み取り専用)。 *my\_mc* ムービークリップのフレームの総数を返します。

## 例

次のコードは、mySWF.swf をレベル 1 にロードし、25 フレーム後にロードされているかどうかをチェックします。

```
loadMovieNum("mySWF.swf", 1);

// メインタイムラインの 25 フレーム後ろ
if (_level1._framesloaded >= _level1._totalframes) {
    tellTarget("_level1") {
        gotoAndStop("myLabel");
    }
} else {
    // ループ ...
}
```

# **\_visible**

## **使用できるバージョン**

Flash Lite 1.0

## **シンタックス**

*my\_mc*:\_visible

## **説明**

プロパティ。*my\_mc* に指定した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。非表示のムービークリップ (`_visible` プロパティを `false` に設定) は使用不可になります。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。この方法で明示的に非表示にされた場合以外はムービークリップは表示されます。

## **例**

次のコードは、ユーザーが **3** のキーを押したときに *my\_mc* ムービークリップを無効にし、ユーザーが **4** のキーを押したときに有効にします。

```
on(keyPress "3") {  
    my_mc:_visible = 0;  
}  
  
on(keyPress "4") {  
    my_mc:_visible = 1;  
}
```

# **\_width**

## **使用できるバージョン**

Flash Lite 1.0

## **シンタックス**

*my\_mc*:\_width

## **説明**

プロパティ。ピクセル単位で示したムービークリップの幅です。

## **例**

次の例では、ユーザーが **5** のキーを押したときにムービークリップの `width` プロパティを設定します。

```
on(keyPress "5") {  
    my_mc:_width = 10;  
}
```

## **\_x**

### 使用できるバージョン

Flash Lite 1.0

### シンタックス

`my_mc:_x`

### 説明

プロパティ。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 (`my_mc`) の x 座標を設定する整数です。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。

ムービークリップが、変形された別のムービークリップ内にある場合、そのムービークリップはそれを包含するムービークリップのローカル座標系に属します。たとえば、反時計回りに 90 度回転されたムービークリップに含まれている子ムービークリップは、反時計回りに 90 度回転され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。

### 例

次の例では、ユーザーが 6 のキーを押したときに `my_mc` ムービークリップの水平位置を変更します。

```
on(keyPress "6") {  
    my_mc:_x = 10;  
}
```

### 関連項目

[\\_xscale](#)、[\\_y](#)、[\\_yscale](#)

## **\_xscale**

### 使用できるバージョン

Flash Lite 1.0

### シンタックス

`my_mc:_xscale`

### 説明

プロパティ。ムービークリップの基準点から適用されるようにムービークリップの水平スケール (%) を設定します。デフォルトの基準点は (0, 0) です。

`_x` および `_y` プロパティの設定はピクセル数で定義されるため、ローカル座標系を伸縮するとこれらのプロパティに影響します。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、ムービークリップ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、ユーザーが7のキーを押したときに `my_mc` ムービークリップの水平スケールを変更します。

```
on(keyPress "7") {  
    my_mc:_xscale = 10;  
}
```

## 関連項目

[\\_x](#)、[\\_y](#)、[\\_yscale](#)

# \_y

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

`my_mc:_y`

## 説明

プロパティ。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 (`my_mc`) の `y` 座標を設定する整数です。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。

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。たとえば、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。

## 例

次の例では、ユーザーが1のキーを押したときに、`my_mc` ムービークリップの `y` 座標を親クリップの (0, 0) 座標の10ピクセル下に設定します。

```
on(keyPress "1") {  
    my_mc:_y = 10;  
}
```

## 関連項目

[\\_x](#)、[\\_xscale](#)、[\\_yscale](#)

# \_yscale

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
my_mc:_yscale
```

## 説明

プロパティ。ムービークリップの基準点から適用されるようにムービークリップの垂直スケール(パーセント)を設定します。デフォルトの基準点は(0,0)です。

\_x および \_y プロパティの設定はピクセル数で定義されるため、ローカル座標系を伸縮するとこれらのプロパティに影響します。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、拡大 / 縮小率が 100% であるときの半分のピクセル数になります。

## 例

次の例では、ユーザーが 1 のキーを押したときに、my\_mc ムービークリップの垂直スケールを 10% に変更します。

```
on(keyPress "1") {  
    my_mc:_yscale = 10;  
}
```

## 関連項目

[\\_x](#)、[\\_xscale](#)、[\\_y](#)

この項では、アドビ システムズ社の Macromedia Flash Lite 1.x ActionScript ステートメントのシンタックスと使用方法について説明します。ActionScript ステートメントは、アクションを実行または指定する言語エレメントです。下表にステートメントをまとめます。

ステートメント	説明
<code>break</code>	ループ本体の残りの部分をスキップし、ループ処理を停止し、ループステートメントの次のステートメントを実行するように Flash Lite に指示します。
<code>case</code>	<code>switch</code> ステートメントの条件を定義します。 <code>case</code> キーワードの次の <i>expression</i> パラメータが、 <code>switch</code> ステートメントの <i>expression</i> と等しい場合に、 <i>statements</i> パラメータのステートメントが実行されます。
<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>do..while</code>	ステートメントを実行し、次に条件が <code>true</code> の間、ループ内の条件を評価します。
<code>else</code>	<code>if</code> ステートメントの条件が <code>false</code> だと評価された場合に、実行す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> ステートメントが <code>false</code> 値を返したときに実行するステートメントを指定します。
<code>for</code>	<i>init</i> (初期化) 式を 1 回評価し、次にループシーケンスを開始し、それによって <i>condition</i> が <code>true</code> だと評価されている間、 <i>statement</i> を実行し、次の式を評価します。
<code>if</code>	条件を評価して、SWF ファイルの次のアクションを決定します。条件が <code>true</code> の場合は、条件に続く中括弧 ( <code>{}</code> ) 内のステートメントが実行されます。条件が <code>false</code> 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。
<code>switch</code>	<code>if</code> ステートメントと同様に、 <code>switch</code> ステートメントは条件をテストし、条件が <code>true</code> だと評価された場合にステートメントを実行します。
<code>while</code>	式をテストし、式が <code>true</code> の間、1 つのステートメントまたは一連のステートメントをループ内で繰り返し実行します。

# break

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
break
```

## パラメータ

なし

## 説明

ステートメント。ループ内 (for、do..while、または while) 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 ステートメントは、ループ本体の残りの部分をスキップし、ループ処理を停止し、ループステートメントの次のステートメントを実行するように Flash Lite に指示します。break ステートメントを使用すると、ActionScript インタプリタがその 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 ステートメントの次にある最初のステートメントにジャンプします。このステートメントは、ネストされている一連のループを強制終了するときに使用します。

## 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。

```
i = 0;
while (true) {
    if (i >= 100) {
        break;
    }
    i++;
}
```

## 関連項目

[case](#)、[do..while](#)、[for](#)、[switch](#)、[while](#)

# case

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
case expression: statements
```

## パラメータ

*expression* 任意の式。

*statements* 任意のステートメント。

## 説明

ステートメント; switch ステートメントの条件を定義します。case キーワードの次の *expression* パラメータが、switch ステートメントの *expression* と等しい場合に、*statements* パラメータ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、コードはコンパイルされません。

## 例

次の例では、myNum パラメータが1だと評価された場合は、case 1 の後にある `trace()` ステートメントが実行され、myNum パラメータが2だと評価された場合は、case 2 の後にある `trace()` ステートメントが実行されます。以下同様です。いずれの case 式も number 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()` が実行されます。

```
switch (myNum) {
    case 1:
        trace ("case 1 tested true");
        break;
    case 2:
        trace ("case 2 tested true");
        break;
    case 3:
        trace ("case 3 tested true");
        break;
    default:
        trace ("no case tested true")
}
```

次の例では、最初の case グループに `break` がないため、数値が1の場合は、A と B の両方が [ 出力 ] パネルに表示されます。

```
switch (myNum) {
    case 1:
        trace ("A");
```

```
case 2:
    trace ("B");
    break;
default:
    trace ("D")
}
```

## 関連項目

[switch](#)

# continue

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

continue

## パラメータ

なし

## 説明

ステートメント。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。

- 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。
- 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、条件が再度評価されます。
- 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、for ループの再初期化式の評価にジャンプします。

## 例

次の while ループで continue を使用すると、Flash Lite 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。

```
i = 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 Lite 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、条件が再度評価されます。

```
i = 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 Lite 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、`i` を 3 で割った余り (余剰) 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。

```
for (i = 0; i < 10; i++) {  
    if (i % 3 == 0) {  
        continue;  
    }  
    trace(i);  
}
```

## 関連項目

[do..while](#)、[for](#)、[while](#)

# do..while

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
do {  
    statement(s)  
} while (condition)
```

## パラメータ

*statement(s)* *condition* パラメータの評価が `true` である間、実行されるステートメント。  
*condition* 評価する条件。

## 説明

ステートメント。ステートメントを実行してから、条件を評価し、条件が `true` の間ループ処理します。

## 例

次の例では、変数の値が10未満の間、インデックス変数がインクリメントされます。

```
i = 0;
do {
    //trace (i);    // 出力 : 0、1、2、3、4、5、6、7、8、9
    i++;
} while (i<10);
```

## 関連項目

[break](#)、[continue](#)、[for](#)、[while](#)

# else

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
if (condition){
    t-statement(s);
} else {
    f-statement(s);
}
```

## パラメータ

*condition* true または false に評価される式。

*t-statement(s)* 条件の評価が true である場合に実行される指示。

*f-statement(s)* 条件の評価が false である場合に代わって実行される一連の指示。

## 説明

ステートメント。if ステートメントの条件が false であると評価されたときに実行されるステートメントを指定します。

## 例

次の例では、条件付きの else ステートメントが使用されます。実際の例では、条件に基づいてアクションを実行するためのコードが含まれます。

```
currentHighestDepth = 1;
if (currentHighestDepth == 2) {
    //trace ("currentHighestDepth is 2");
} else {
    //trace ("currentHightestDepth is not 2");
}
```

## 関連項目

[if](#)

# else if

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
if (condition){  
    statement(s);  
} else if (condition){  
    statement(s);  
}
```

## パラメータ

*condition* true または false に評価される式。

*statement(s)* if ステートメントで指定された条件が false で、なおかつ else if ステートメントの条件が true である場合に実行される一連のステートメント。

## 説明

ステートメント。条件を評価し、最初の if ステートメントの条件から false 値を返されたときに実行するステートメントを指定します。else if 条件が true 値を返した場合は、Flash インタープリタが else if 条件に続く中括弧({})内のステートメントを実行します。else if 条件が false である場合は、中括弧内のステートメントをスキップし、中括弧の後のステートメントを実行します。スクリプト内に分岐処理を作成するには else if ステートメントを使用します。

## 例

次の例では、else if ステートメントを使用して、オブジェクトの各辺が特定の境界内にあるかどうかをチェックします。

```
person_mc.xPos = 100;  
leftBound = 0;  
rightBound = 100;  
  
if (person_mc.xPos <= leftBound) {  
    //trace ("Clip is to the far left");  
} else if (person_mc.xPos >= rightBound) {  
    //trace ("Clip is to the far right");  
} else {  
    //trace ("Your clip is somewhere in between");  
}
```

## 関連項目

[if](#)

# for

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
for ( init; condition; next ) {  
    statement(s);  
}
```

## 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。

*condition* true または false に評価される式。条件は、各ループの実行前に評価されます。条件の評価が false の場合はループから抜けます。

*next* 各ループ実行後に評価される式。通常は、インクリメント (++) またはデクリメント (--) 演算子を使用する代入式です。

*statement(s)* ループ内で実行される指示。

## 説明

ステートメント。ループ構造体が *init* (初期化) 式を 1 回評価し、次にループシーケンスを開始し、それによって、*condition* が true だと評価されている間、*statement* を実行し、次の式を評価します。プロパティの中には、for または for..in ステートメント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。

## 例

次の例では、for ループを使用して 1 から 100 の数値を加算します。

```
sum = 0;  
for ( i = 1; i <= 100; i++ ) {  
    sum = sum + i;  
}
```

## 関連項目

[++ \(インクリメント\)](#)、[-- \(デクリメント\)](#)、[do..while](#)、[while](#)

# if

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
if (condition) {  
    statement(s);  
}
```

## パラメータ

*condition* true または false に評価される式。

*statement(s)* 条件の評価が true である場合に実行される指示。

## 説明

ステートメント。SWF ファイルの次のアクションを決定するために条件を評価します。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 ステートメントを使用します。

## 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" が格納されているかどうかを確認します。含まれている場合は、`play()` 関数が実行されます。

```
if(name eq "Erica"){  
    play();  
}
```

# switch

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
switch (expression){  
    caseClause:  
    [defaultClause:]  
}
```

## パラメータ

*expression* 任意の式。

*caseClause* case キーワードとそれに続く式、コロン、およびステートメント群。ステートメント群は、式と `switch` の *expression* パラメータが一致した場合に実行されます。

*defaultClause* オプションの `default` キーワードとそれに続くステートメント群。ステートメント群は、いずれの `case` 式も `switch` の *expression* パラメータと一致しない場合に実行されます。

## 説明

ステートメント。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` であると評価された場合にステートメントを実行します。

Switch ステートメントには、`default` と呼ばれる代替オプションが含まれます。他のすべての `case` 式が `true` でない場合は、`default` ステートメントが実行されます。

## 例

次の例では、`myNum` パラメータが 1 と評価された場合、`case 1` の後にある `trace()` ステートメントが実行されます。`myNum` パラメータが 2 と評価された場合、`case 2` の後にある `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 `myNum` 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()` が実行されます。

```
switch (myNum) {
    case 1:
        trace ("case 1 tested true");
        break;
    case 2:
        trace ("case 2 tested true");
        break;
    case 3:
        trace ("case 3 tested true");
        break;
    default:
        trace ("no case tested true")
}
```

次の例では、最初の `case` グループに `break` がないため、数値が 1 の場合は、A と B の両方が [ 出力 ] パネルに表示されます。

```
switch (myNum) {
    case 1:
        trace ("A");
    case 2:
        trace ("B");
        break;
    default:
        trace ("D")
}
```

## 関連項目

[case](#)

# while

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
while(condition) {  
    statement(s);  
}
```

## パラメータ

*condition* while ステートメントが実行されるたびに評価される式。

*statement(s)* 条件の評価が true である場合に実行される指示。

## 説明

ステートメント。式をテストし、式が true の間、1つのステートメントまたは一連のステートメントをループ内で繰り返し実行します。

ステートメントブロックを実行する前に条件をテストし、テストが true を返した場合にステートメントブロックを実行します。条件が false の場合は、ステートメントブロックがスキップされ、while ステートメントのステートメントブロックの後にある最初のステートメントが実行されます。

一般にループ処理は、カウンタ変数が指定値より小さいときに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を加算します。指定された値に達すると、条件は true でなくなり、ループは終了します。

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。各反復の初めに *condition* が検査されます。

1. 式 *condition* が評価されます。
2. *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。  
それ以外は、while ステートメントが完了され、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。
3. ステートメントブロック *statement(s)* を実行します。
4. 手順 1 に進みます。

## 例

次の例では、インデックス変数 *i* の値が 10 未満の間、ループが実行されます。

```
i = 0;  
while(i < 10) {  
    trace("i = " + i); // Output: 1,2,3,4,5,6,7,8,9  
}
```

## 関連項目

[continue](#)、[do..while](#)、[for](#)



この項では、アドビ システムズ社の Macromedia Flash Lite 1.x ActionScript の演算子のシンタックスと使用方法について説明します。各項目はアルファベット順に示します。演算子が記号の場合は、テキストの説明のアルファベット順に示します。

この項で説明する演算子を下表にまとめます。

演算子	説明
<code>add</code> ( ストリング連結 )	複数のストリングを連結 ( 結合 ) します。
<code>+=</code> ( 加算後代入 )	<code>expression1</code> に <code>expression1 + expression2</code> の値を代入します。
<code>and</code>	論理 AND 演算を行います。
<code>=</code> ( 代入 )	<code>expression2</code> の値 ( 右側のオペランド ) を <code>expression1</code> の変数またはプロパティに代入します。
<code>/*</code> ( ブロックコメント )	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 ( <code>/*</code> ) と終了コメントタグ ( <code>*/</code> ) 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。
<code>,</code> ( カンマ )	<code>expression1</code> を評価し、次に <code>expression2</code> を評価して、 <code>expression2</code> の値を返します。
<code>//</code> ( コメント )	スクリプトコメントの先頭を示します。コメントブロック区切り記号 ( <code>//</code> ) 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。
<code>?:</code> ( 条件演算子 )	Flash Lite に <code>expression1</code> を評価するように指示し、 <code>expression1</code> の値が <code>true</code> の場合は、演算子が <code>expression2</code> の値を返します。それ以外の場合は、 <code>expression3</code> の値を返します。
<code>--</code> ( デクリメント )	<code>expression</code> から 1 を減算します。プリデクリメント形式の演算子 ( <code>--expression</code> ) は、 <code>expression</code> から 1 を減算し、結果を数値で返します。ポストデクリメント形式の演算子 ( <code>expression--</code> ) は、 <code>expression</code> から 1 を減算し、 <code>expression</code> の初期値 ( 減算前の値 ) を返します。
<code>/</code> ( 除算 )	<code>expression1</code> を <code>expression2</code> で除算します。
<code>/=</code> ( 除算後代入 )	<code>expression1</code> に <code>expression1 / expression2</code> の値を代入します。

演算子	説明
. (ドット)	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。
++ (インクリメント)	<i>expression</i> に 1 を加算します。 <i>expression</i> 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 ( <i>++expression</i> ) は、 <i>expression</i> に 1 を加算し、結果を数値で返します。ポストインクリメント形式の演算子 ( <i>expression++</i> ) は、 <i>expression</i> に 1 を加算し、 <i>expression</i> の初期値 (加算前の値) を返します。
&& (論理積 (AND))	その結果は、左の式をブール (論理) 値で評価した値が <b>false</b> であれば左の式の値であり、それ以外は右の式の値です。
! (否定 (NOT))	変数または式のブール値を反転します。 <i>expression</i> が変数で、その絶対値または変換された値が <b>true</b> である場合、 <i>!expression</i> の値は <b>false</b> になります。式 <i>x &amp;&amp; y</i> の評価が <b>false</b> である場合、式 <i>!(x &amp;&amp; y)</i> の評価は <b>true</b> です。
(論理和 (OR))	<i>expression1</i> と <i>expression2</i> を評価します。いずれかまたは両方の式の評価が <b>true</b> の場合は、結果が <b>true</b> になります。両方の式の評価が <b>false</b> の場合にのみ、結果が <b>false</b> 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 <b>true</b> であれば、結果は <b>true</b> です。
% (剰余)	<i>expression1</i> を <i>expression2</i> で割ったときの剰余を計算します。 <i>expression</i> オペランドが数値以外の場合は、剰余演算子によって数値に変換されます。
%= (剰余を代入)	<i>expression1</i> に <i>expression1 % expression2</i> の値を代入します。
*= (乗算後代入)	<i>expression1</i> に <i>expression1 * expression2</i> の値を代入します。
* (乗算)	2 つの数値式の乗算を行います。
+ (数値加算)	数値式を追加します。
== (数値等価)	等価性をテストします。 <i>expression1</i> が <i>expression2</i> と等しい場合は、結果は <b>true</b> です。
> (より大きい - 数値)	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決定します。より大きい場合、演算子が <b>true</b> を返します。 <i>expression1</i> が <i>expression2</i> より小さいか等しい場合は、 <b>false</b> を返します。
>= (より大きいか等しい - 数値)	2 つの式を比較し、 <i>expression1</i> が <i>expression2</i> より大きいか等しい ( <b>true</b> )、または <i>expression1</i> が <i>expression2</i> より小さい ( <b>false</b> ) のいずれであるかを決定します。
<> (数値不等価)	不等価性をテストします。 <i>expression1</i> が <i>expression2</i> と等しい場合は、結果は <b>false</b> です。

演算子	説明
< (より小さい - 数値)	2つの式を比較し、 <i>expression1</i> が <i>expression2</i> より小さいかどうか決定します。より小さい場合、演算子が true を返します。 <i>expression1</i> が <i>expression2</i> より大きいか等しい場合は、false を返します。
<= (より小さいか等しい - 数値)	2つの式を比較し、 <i>expression1</i> が <i>expression2</i> より小さいまたは等しいかを決定します。小さいか等しい場合は、演算子が true を返し、それ以外の場合は false を返します。
() (括弧)	1つ以上のパラメータをグループ化するか、複数の式を順番に評価します。または、1つ以上のパラメータを囲み、結果をパラメータとして括弧の外側の関数に渡します。
" " (ストリング区切り記号)	引用符で前後を囲んだ 0 個以上の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。
eq (ストリング等価)	2つの式の等価性を比較し、 <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。
gt (より大きい - ストリング)	<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。
ge (より大きいか等しい - ストリング)	<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> より大きいか等しい場合は、true 値を返します。それ以外の場合は false を返します。
ne (ストリング不等価)	<i>expression1</i> のストリング表現を <i>expression2</i> と比較し、 <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。
lt (より小さい - ストリング)	<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> より小さい場合は、true 値を返します。それ以外の場合は false を返します。
le (より小さいか等しい - ストリング)	<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> より小さいか等しい場合は、true 値を返します。それ以外の場合は false を返します。
- (減算)	符号反転または減算に使用します。
-= (減算後代入)	<i>expression1</i> に <i>expression1</i> - <i>expression2</i> の値を代入します。

# add ( ストリング連結 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
string1 add string2
```

## パラメータ

*string1*、*string2* ストリング。

## 説明

演算子。複数のストリングを連結 ( 結合 ) します。

## 例

次の例では、2つのストリング値を結合して catalog というストリングにします。

```
conStr = "cat" add "alog";  
trace (conStr);// 出力 : catalog
```

## 関連項目

[+ \( 数値加算 \)](#)

# += ( 加算後代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 += expression2
```

## パラメータ

*expression1*、*expression2* 数値またはストリング。

## 説明

演算子 ( 算術複合代入 )。 *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;
```

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。

## 例

次の例では、加算して代入(+=) 演算子を使用して、x の値に y の値を加算します。

```
x = 5;
y = 10;
x += y;
trace(x);// 出力 : 15
```

## 関連項目

[+ \( 数値加算 \)](#)

# and

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
condition1 and condition2
```

## パラメータ

*condition1*, *condition2* true または false に評価される条件または式。

## 説明

演算子。論理 AND 演算を行います。

## 例

次の例では、and 演算子を使用して、プレイヤーのゲームの勝敗をテストします。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。

```
turns = 2;
score = 77;
winner = (turns <= 3) and (score >= 75);
if (winner) {
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
// 出力 : You Win the Game!
```

## 関連項目

[&& \( 論理積 \(AND\)\)](#)

# = ( 代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 = expression2
```

## パラメータ

*expression1* 変数またはプロパティ。

*expression2* 値。

## 説明

演算子。 *expression2* の値 ( 右側のオペランド ) を *expression1* の変数またはプロパティに代入します。

## 例

次の例では、代入 (=) 演算子を使用して数値を変数 `weight` に代入します。

```
weight = 5;
```

次の例では、代入 (=) 演算子を使用してストリング値を変数 `greeting` に代入します。

```
greeting = "Hello, " and personName;
```

# /\* ( ブロックコメント )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

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

## パラメータ

*comment* 任意の文字。

## 説明

コメント区切り記号。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。

単一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。連続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、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。

## 関連項目

[// \(コメント\)](#)

# , (カンマ)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1*, *expression2*

## パラメータ

*expression1*, *expression2* 数値に評価される数値または式。

## 説明

演算子。 *expression1* を評価し、次に *expression2* を評価し、 *expression2* の値を返します。

## 例

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。

```
v = 0;
v = 4, 5, 6;
trace(v); // 出力 : 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。

```
v = 0;
v = (4, 5, 6);
trace(v); // 出力 : 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。

```
v = 0;
z = 0;
v = v + 4 , z++, v + 6;
trace(v); // 出力 : 4
trace(z); // 出力 : 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。

```
v = 0;
z = 0;
v = (v + 4 , z++, v + 6);
trace(v); // 出力 : 6
trace(z); // 出力 : 1
```

## 関連項目

[for, \( \) \(括弧\)](#)

# // (コメント)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
// comment
```

## パラメータ

*comment* 任意の文字。

## 説明

コメントブロック区切り記号。スクリプトコメントの先頭を示します。コメントブロック区切り記号 (//) 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。

## 例

次の例では、コメントブロック区切り記号を使用して第1、第3、第5、第7の行をコメントとして識別します。

```
// ボールムービークリップの X 座標を記録
ballX = ball._x;
// ボールムービークリップの Y 座標を記録
ballY = ball._y;
// バットムービークリップの X 座標を記録
batX = bat._x;
// バットムービークリップの Y 座標を記録
batY = bat._y;
```

## 関連項目

[/\\* \( ブロックコメント \)](#)

# ?: ( 条件演算子 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 ? expression2 : expression3
```

## パラメータ

*expression1* ブール値と評価される式。通常は  $x < 5$  などの比較式。

*expression2*、*expression3* 任意のタイプの値。

## 説明

演算子。*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。

## 例

次の例では、*expression1* の評価が true であるために、変数 *x* の値を変数 *z* に代入します。

```
x = 5;
y = 10;
z = (x < 6) ? x : y;
trace(z); // 出力 : 5
```

# -- (デクリメント)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*--expression*

*expression--*

## パラメータ

なし

## 説明

演算子 ( 算術演算 )。 *expression* から 1 を引くプリデクリメントとポストデクリメントの単項演算子。プリデクリメント形式の演算子 (*--expression*) は、 *expression* から 1 を減算し、結果を数値で返します。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。

## 例

次の例では、プリデクリメント形式の演算子が *aWidth* を 2 にデクリメント ( $aWidth - 1 = 2$ ) して、結果を *bWidth* として返します。

```
aWidth = 3;
bWidth = --aWidth;
// bWidth 値は 2
```

次の例では、ポストデクリメント形式の演算子が *aWidth* を 2 にデクリメント ( $aWidth - 1 = 2$ ) して、元の値の *aWidth* を結果 *bWidth* として返します。

```
aWidth = 3;
bWidth = aWidth--;
// bWidth 値は 3
```

# /( 除算 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1 / expression2*

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 (算術演算)。 *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点数です。

## 例

次のステートメントでは、浮動小数点数 22.0 を 7.0 で除算して、[ 出力 ] パネルに結果を表示します。

```
trace(22.0 / 7.0);
```

結果は 3.1429 で、浮動小数点数です。

# /= ( 除算後代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 /= expression2
```

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 (算術複合代入); *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
```

## 例

次の例では、変数と数値と共に /= 演算子を使用します。

```
x = 10;  
y = 2;  
x /= y;  
// 現在、式 x には値 5 が格納されています。
```

# . (ドット)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*instancename.variable*

*instancename.childinstance.variable*

## パラメータ

*instancename* ムービークリップのインスタンス名

*childinstance* 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

*variable* 指定したムービークリップのインスタンス名のタイムライン上の変数。

## 説明

演算子。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。

## 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
```

これは、以下のスラッシュシンタックス表記と同じです。

```
/person_mc:hairColor
```

## 関連項目

[/\(スラッシュ\)](#)

# ++ (インクリメント)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*++expression*

*expression++*

## パラメータ

なし

## 説明

演算子 (算術演算)。 *expression* に1を加えるプリインクリメントとポストインクリメントの単項演算子。 *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、結果を数値で返します。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。

## 例

次の例では、 ++ をポストインクリメントの演算子として使用し、 while ループを 5 回実行します。

```
i = 0;
while (i++ < 5){
    trace("this is execution " + i);
}
```

次の例では、 ++ をプリインクリメントの演算子として使用します。

```
a = "";
i = 0;
while (i < 10) {
    a = a add (++i) add ",";
}
trace(a);// 出力 : 1、2、3、4、5、6、7、8、9、10、
```

このスクリプトを実行すると、 [出力] パネルに次の結果が表示されます。

1,2,3,4,5,6,7,8,9,10,

次の例では、 ++ をポストインクリメントの演算子として使用します。

```
a = "";
i = 0;
while (i < 10) {
    a = a add (i++) add ",";
}
trace(a);// 出力 : 0、1、2、3、4、5、6、7、8、9、
```

このスクリプトを実行すると、 [出力] パネルに次の結果が表示されます。

0,1,2,3,4,5,6,7,8,9,

# && (論理積 (AND))

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* && *expression2*

## パラメータ

*expression1*、*expression2* ブール値に変換されるブール値または式。

## 説明

演算子 (論理)。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。その結果は、左の式をブール (論理) 値で評価した値が **false** であれば左の式の値であり、それ以外は右の式の値です。

次の例では、(&&) 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。

```
turns = 2;
score = 77;
winner = (turns <= 3) && (score >= 75);
if (winner) {
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
```

次の例では、架空の x 座標が範囲内かどうかテストします。

```
xPos = 50;
if (xPos >= 20 && xPos <= 80) {
    trace("the xPos is in between 20 and 80");
}
```

# !(否定 (NOT))

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

!*expression*

## パラメータ

なし

## 説明

演算子 (論理)。変数または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` を返します。

`!false` は `true` を返します。

## 例

次の例では、変数 `happy` が `false` に設定されています。 `if` 文が条件 `!happy` を評価し、その結果が `true` である場合、 `trace()` 関数は [出力] パネルにストリングを表示します。

```
happy = false;
if (!happy) {
    trace("don't worry, be happy");
}
```

# ||(論理和 (OR))

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* || *expression2*

## パラメータ

*expression1*、*expression2* ブール値に変換されるブール値または式。

## 説明

演算子 (論理)。 *expression1* と *expression2* を評価します。いずれかまたは両方の式の評価が `true` の場合は、結果が `true` になります。両方の式の評価が `false` の場合にのみ、結果が `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。

ブール式以外に論理和 (OR) 演算子を使用すると、左側の式が評価されます。式を `true` に変換できる場合は、結果は左の式の値になります。変換できない場合は、右側の式が評価され、結果はその式の値になります。

## 例

シンタックス1: 次の例では、if 文で || 演算子を使用しています。2 番目の式の評価は true であるため、最終結果は true です。

```
theMinimum = 10;
theMaximum = 250;
start = false;
if (theMinimum > 25 || theMaximum > 200 || start){
    trace("the logical OR test passed");
}
```

# % ( 剰余 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* % *expression2*

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 (算術演算); *expression1* を *expression2* で割ったときの剰余を計算します。*expression* オペランドが数値以外の場合は、剰余演算子によって数値に変換されます。*expression* は数値または数値に変換される文字列です。

Flash Lite 1.0 または 1.1 をターゲットとする場合は、Flash コンパイラがパブリッシュされた SWF ファイルの % 演算子を次の式を使用して拡張します。

*expression1* - int(*expression1*/*expression2*) \* *expression2*

この近似のパフォーマンスは、剰余演算子がネイティブでサポートされる Flash Player のバージョンと比較して、速度や正確さが劣る場合があります。

## 例

次のコードでは、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); // 出力 : 2
trace (4.3 % 2.1); // 出力 : 0.0999...
```

# %= ( 剰余を代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* %= *expression2*

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 (算術複合代入); *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
```

## 例

次の例では、値 4 を変数 x に代入します。

```
x = 14;  
y = 5;  
trace(x %= y); // 出力 : 4
```

## 関連項目

[% \( 剰余 \)](#)

# \*= ( 乗算後代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* \*= *expression2*

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 (算術複合代入)。 *expression1* に *expression1 \* expression2* の値を代入します。

たとえば、次の2つの式は同じ結果になります。

```
x *= y
x = x * y
```

## 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。

```
x = 5;
y = 10;
trace(x *= y); // 出力 : 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。

```
i = 5;
x = 4 - 6;
y = i + 2;
trace(x *= y); // 出力 : -14
```

# \* (乗算)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 * expression2
```

## パラメータ

*expression1*、*expression2* 数値式。

## 説明

演算子 (算術演算)。2つの数値式を乗算します。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点数であれば、その積は浮動小数点数です。

## 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。

```
2 * 3
```

結果は 6 で、整数です。

シンタックス2: 次のステートメントは、浮動小数点数 2.0 と 3.1416 を乗算します。

```
2.0 * 3.1416
```

結果は 6.2832 で、浮動小数点数です。

# + ( 数値加算 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* + *expression2*

## パラメータ

*expression1*、*expression2* 数値。

## 説明

演算子。数値式を追加します。+ は数値演算子としてのみ使用します。string連結には使用できません。

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点数の場合、合計は浮動小数点数になります。

## 例

次の例では、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 );
```

次の例では、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 );
```

## 関連項目

[add \( string連結 \)](#)

# == ( 数値等価 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* == *expression2*

## パラメータ

*expression1*、*expression2* 数値、ブール値、または変数。

## 説明

演算子 ( 比較 )。等価性をテストします。<> 演算子の正反対です。 *expression1* が *expression2* に等しい場合、結果は true です。<> 演算子と同様に *等価* の定義は比較するデータ型によって異なります。

- 数値とブール値は値で比較されます。
- 変数は参照で比較されます。

## 例

次の例は、true と false の戻り値を示します。

```
trees = 7;
bushes = "7";
shrubs = "seven";

trace (trees == "7");// 出力 : 1(true)
trace (trees == bushes);// 出力 : 1(true)
trace (trees == shrubs);// 出力 : 0(false)
```

## 関連項目

[eq \( スtring等価 \)](#)

# > ( より大きい - 数値 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 > expression2
```

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

演算子 ( 比較 ); 2 つの式を比較し、*expression1* が *expression2* より大きいかどうかを決定します。より大きい場合、この演算子は 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。

## 例

次の例では、数値の比較で true と false が返される場合の例を示します。

```
trace(3.14 > 2);// 出力 : 1(true)
trace(1 > 2);// 出力 : 0(false)
```

## 関連項目

[gt \( より大きい - String \)](#)

# >= ( より大きい か等しい - 数値 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* >= *expression2*

## パラメータ

*expression1*、*expression2* 整数または浮動小数。

## 説明

演算子 ( 比較 )。2 つの式を比較し、*expression1* が *expression2* より大きい か等しい (true)、または *expression1* が *expression2* より小さい (false) のいずれであるかを評価します。

## 例

次の例は、true と false の結果を示します。

```
trace(3.14 >= 2); // 出力 : 1(true)
trace(3.14 >= 4); // 出力 : 0(false)
```

## 関連項目

[ge \( より大きい か等しい - スtring \)](#)

# <> ( 数値不等価 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* <> *expression2*

## パラメータ

*expression1*、*expression2* 数値、ブール値、または変数。

## 説明

演算子 ( 比較 )。不等価をテストします。等価(==) 演算子の正反対の演算子です。*expression1* が *expression2* に等しい場合、結果は false です。等価(==) 演算子の場合と同様に、" 等価 " の定義は比較されるデータ型により異なります。

- 数値とブール値は値で比較されます。
- 変数は参照で比較されます。

## 例

次の例は、true と false の戻り値を示します。

```
trees = 7;
B = "7";

trace(trees <> 3);// 出力 : 1(true)
trace(trees <> B);// 出力 : 0(false)
```

## 関連項目

[ne \( ストリング不等価 \)](#)

# < ( より小さい - 数値 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 < expression2
```

## パラメータ

*expression1*、*expression2* 数値。

## 説明

演算子 ( 比較 ); 2つの式を比較し、*expression1* が *expression2* より小さいかどうかを決定します。より小さい場合、この演算子は true を返します。*expression1* が *expression2* より大きい場合は、false を返します。< ( より小さい ) 演算子は数値演算子です。

## 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 ( 3 < 10 );// 出力 : 1(true)

trace ( 10 < 3 );// 出力 : 0(false)
```

## 関連項目

[lt \( より小さい - ストリング \)](#)

# <= ( より小さいか等しい - 数値 )

Flash Lite 1.0

## シンタックス

```
expression1 <= expression2
```

## パラメータ

*expression1*、*expression2* 数値。

## 説明

演算子 ( 比較 )。2つの式を比較し、*expression1*が*expression2*より小さいか等しいかどうかを判断します。小さいか等しい場合は、演算子が true を返し、それ以外の場合は false を返します。この演算子が使用できるのは、数値の比較の場合だけです。

## 例

次の例では、数値の比較で true と false が返される場合の例を示します。

```
trace(5 <= 10);// 出力 : 1(true)
trace(2 <= 2);// 出力 : 1(true)
trace (10 <= 3);// 出力 : 0(false)
```

## 関連項目

[!e \( より小さいか等しい - ストリング \)](#)

# () ( 括弧 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
( expression1 [, expression2] )
( expression1, expression2 )
```

*expression1*、*expression2* 数値、ストリング、変数、またはテキスト。

*parameter1*、...、*parameterN* 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。

## 説明

演算子。1つ以上のパラメータをグループ化するか、複数の式を順番に評価します。または、1つ以上のパラメータを囲み、結果をパラメータとして括弧の外側の関数に渡します。

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、括弧内の式が最初に評価されます。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。

## 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)); // 45 を表示します
trace(2 + (3 * (4 + 5))); // // 29 を表示します
trace(2 + (3 * 4) + 5); // 19 を表示します
```

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 (各式の値がログファイルに書き込まれます)。

```
trace((2 + 3) * (4 + 5)); // 45 を書き込みます
trace(2 + (3 * (4 + 5))); // 29 を書き込みます
trace(2 + (3 * 4) + 5); // 19 を書き込みます
```

# " " ( ストリング区切り記号 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
"text"
```

## パラメータ

*text* 0個以上の文字。

## 説明

ストリング区切り記号。引用符で前後を囲んだ0個以上の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。

## 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。

```
yourGuess = "Prince Edward Island";

on(release){
    province = yourGuess;
    trace(province);// 出力 : Prince Edward Island
}
```

# eq ( ストリング等価 )

使用できるバージョン

Flash Lite 1.0

シンタックス

```
expression1 eq expression2
```

パラメータ

*expression1*、*expression2* 数値、ストリング、または変数。

説明

比較演算子。2つの式が等しいかどうかを比較し、*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、それ以外は false を返します。

例

次の例は、true と false の結果を示します。

```
word = "persons";  
figure = "55";  
  
trace("persons" eq "people");// 出力 : 0(false)  
trace("persons" eq word);// 出力 : 1(true)  
trace(figure eq 50 + 5);// 出力 : 1(true)  
trace(55.0 eq 55);// 出力 : 1(true)
```

関連項目

[== \( 数値等価 \)](#)

# gt ( より大きい - ストリング )

使用できるバージョン

Flash Lite 1.0

シンタックス

```
expression1 gt expression2
```

パラメータ

*expression1*、*expression2* 数値、ストリング、または変数。

## 説明

演算子 ( 比較 )。 *expression1* の文字列表現を *expression2* の文字列表現と比較し、 *expression1* が *expression2* より大きい場合は true 値を返します。それ以外は、 false 値を返します。文字列はアルファベットの順序を基に比較されます。数字はすべての文字の前になり、大文字はすべての小文字の前になります。

## 例

次の例は、 true と false の結果を示します。

```
animals = "cats";
breeds = 7;

trace ("persons" gt "people");// 出力 : 1(true)
trace ("cats" gt "cattle");// 出力 : 0(false)
trace (animals gt "cats");// 出力 : 0(false)
trace (animals gt "Cats");// 出力 : 1(true)
trace (breeds gt "5");// 出力 : 1(true)
trace (breeds gt 7);// 出力 : 0(false)
```

## 関連項目

[> \( より大きい - 数値 \)](#)

# ge ( より大きいか等しい - 文字列 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 ge expression2
```

## パラメータ

*expression1*、 *expression2* 数値、文字列、または変数。

## 説明

演算子 ( 比較 )。 *expression1* の文字列表現を *expression2* の文字列表現と比較し、 *expression1* が *expression2* より大きいか等しい場合は true 値を返します。それ以外は、 false 値を返します。文字列はアルファベットの順序を基に比較されます。数字はすべての文字の前になり、大文字はすべての小文字の前になります。

## 例

次の例は、true と false の結果を示します。

```
animals = "cats";
breeds = 7;

trace ("cats" ge "cattle");// 出力 : 0(false)
trace (animals ge "cats");// 出力 : 1(true)
trace ("persons" ge "people");// 出力 : 1(true)
trace (animals ge "Cats");// 出力 : 1(true)
trace (breeds ge "5");// 出力 : 1(true)
trace (breeds ge 7);// 出力 : 1(true)
```

## 関連項目

[>= \(より大きいか等しい - 数値\)](#)

# ne ( ストリング不等価 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 ne expression2
```

## パラメータ

*expression1*、*expression2* 数値、ストリング、または変数。

## 説明

演算子 ( 比較 )。 *expression1* のストリング表現を *expression2* と比較し、 *expression1* が *expression2* と等しくない場合は true を返します。それ以外は、 false を返します。

## 例

次の例は、true と false の結果を示します。

```
word = "persons";
figure = "55";

trace ("persons" ne "people");// 出力 : 1(true)
trace ("persons" ne word);// 出力 : 0(false)
trace (figure ne 50 + 5); // 出力 : 0(false)
trace (55.0 ne 55); // 出力 : 0(false)
```

## 関連項目

[<> \(数値不等価\)](#)

# lt (より小さい - ストリング)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 lt expression2
```

## パラメータ

*expression1*、*expression2* 数値、ストリング、または変数。

## 説明

演算子 (比較)。 *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* より小さい場合は true 値を返します。それ以外は、 false 値を返します。ストリングはアルファベットの順序を基に比較されます。数字はすべての文字の前になり、大文字はすべての小文字の前になります。

## 例

次の例は、さまざまなストリングの比較の出力を示しています。最後の行では、ストリングを整数と比較しても lt はエラーを返しません。ActionScript 1.0 シンタックスが整数データ型をストリングに変換して false を返すためです。

```
animals = "cats";  
breeds = 7;  
  
trace ("persons" lt "people");// 出力 : 0(false)  
trace ("cats" lt "cattle");// 出力 : 1(true)  
trace (animals lt "cats");// 出力 : 0(false)  
trace (animals lt "Cats");// 出力 : 0(false)  
trace (breeds lt "5");// 出力 : 0(false)  
trace (breeds lt 7);// 出力 : 0(false)
```

## 関連項目

[< \(より小さい - 数値\)](#)

# le (より小さいか等しい - ストリング)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

*expression1* le *expression2*

## パラメータ

*expression1*、*expression2* 数値、ストリング、または変数。

## 説明

演算子 (比較)。 *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* より小さいか等しい場合は true 値を返します。それ以外は、false 値を返します。ストリングはアルファベットの順序を基に比較されます。数字はすべての文字の前になり、大文字はすべての小文字の前になります。

## 例

次の例は、さまざまなストリングの比較の出力を示しています。

```
animals = "cats";
breeds = 7;

trace ("persons" le "people");// 出力 : 0(false)
trace ("cats" le "cattle");// 出力 : 1(true)
trace (animals le "cats");// 出力 : 1(true)
trace (animals le "Cats");// 出力 : 0(false)
trace (breeds le "5");// 出力 : 0(false)
trace (breeds le 7);// 出力 : 1(true)
```

## 関連項目

[<= \(より小さいか等しい - 数値\)](#)

# - (減算)

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

(符号反転)  $-expression$

(減算)  $expression1 - expression2$

## パラメータ

$expression1$ ,  $expression2$  数値に評価される数値または式。

## 説明

演算子 (算術演算)。符号反転または減算に使用します。

シンタックス 1: 符号反転に使用する場合、数値式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つの数値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点数であれば、その差は浮動小数点数です。

## 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));  
// 出力 : -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);  
// 出力 : 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);  
// 出力 : 1.75
```

結果は、浮動小数点数の 1.75 です。

# -= ( 減算後代入 )

## 使用できるバージョン

Flash Lite 1.0

## シンタックス

```
expression1 -= expression2
```

## パラメータ

*expression1*、*expression2* 数値に評価される数値または式。

## 説明

演算子 ( 算術複合代入 )。 *expression1* に *expression1* - *expression2* の値を代入します。何も値は返されません。

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;
```

ストリング式は数値に変換する必要があります。変換しない場合、-1 が返されます。

## 例

シンタックス 1: 次の例では、-= 演算子を使用して 2 から 3 を減算し、その結果を変数 x に代入します。

```
x = 2;  
y = 3;  
x -= y  
trace(x); // 出力 : -1
```

シンタックス 2: 次の例では、ストリングを数値に変換する方法を示します。

```
x = "2";  
y = "5";  
x -= y;  
trace(x); // 出力 : -3
```



この項では、アドビ システムズ社の Macromedia Flash Lite 1.1 で認識可能なプラットフォーム機能と変数、および `fscommand()` と `fscommand2()` 関数を使用して実行できる Flash Lite コマンドについて説明します。この項で説明する機能は Flash Lite 固有のものであります。

この項の内容を下表にまとめます。

言語エレメント	説明
<code>_capCompoundSound</code>	Flash Lite でコンパウンドサウンドを処理できるかを示します。
<code>_capEmail</code>	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>	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>	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>	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。
<code>_capMMS</code>	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。
<code>_capMP3</code>	デバイスがサウンドデータを MP3 (MPEG Audio Layer 3) オーディオ形式で再生できるかどうかを示します。
<code>_capSMAF</code>	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。
<code>_capSMS</code>	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>	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。

言語エレメント	説明
<code>_cap4WayKeyAS</code>	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。
<code>\$version</code>	Flash Lite のバージョン番号が格納されます。
<code>fscommand()</code>	Launch コマンドを実行するために使用する関数 ( 次の項目を参照 )。
<code>Launch</code>	( <code>fscommand()</code> でサポートされる唯一のコマンド ) SWF ファイルを使用して Flash Lite、または携帯端末やデバイスのオペレーティングシステムなどのホスト環境と通信します。
<code>fscommand2()</code>	この表のコマンドを実行するための関数 ( <code>fscommand()</code> は除きます )。
<code>Escape</code>	任意の文字列をネットワーク転送に安全な形式にエンコードします。
<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>	現在のバッテリーの残量を返します。
<code>GetDateDay</code>	現在日付の日を数値として返します。
<code>GetDateMonth</code>	現在日付の月を数値として返します。
<code>GetDateWeekday</code>	現在の曜日の番号を数値として返します。
<code>GetDateYear</code>	現在の日付の年を表す 4 桁の数値を返します。
<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>	デバイスの一意的識別子 ( 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>GetLanguage</code>	現在デバイスで使用している言語を識別するパラメータを設定します。
<code>GetLocaleLongDate</code>	パラメータを、現在定義されている地域設定に従って書式設定された、長い形式の現在日付を表す文字列に設定します。
<code>GetLocaleShortDate</code>	パラメータを、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を表す文字列 ( 短い形式 ) に設定します。
<code>GetLocaleTime</code>	パラメータを、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を表す文字列に設定します。
<code>GetMaxBatteryLevel</code>	デバイスの最大バッテリー容量を返します。
<code>GetMaxSignalLevel</code>	信号の強さの最大レベルを返します。
<code>GetMaxVolumeLevel</code>	デバイスの最大音量レベルを数値で返します。

言語エレメント	説明
<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>GetNetworkName</code>	パラメータを現在のネットワークの名前に設定します。
<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。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。
<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として返します。
<code>GetTimeHours</code>	現在時刻を 24 時間形式で数値として返します。
<code>GetTimeMinutes</code>	現在時刻の分を数値として返します。
<code>GetTimeSeconds</code>	現在時刻の秒を数値として返します。
<code>GetTimeZoneOffset</code>	パラメータを、ローカルタイムゾーンと世界時 (UTC) の差を表す分に設定します。
<code>GetTotalPlayerMemory</code>	Flash Lite に割り当てられているヒープメモリの総量を KB 単位で返します。
<code>GetVolumeLevel</code>	デバイスの現在の音量レベルを数値として返します。
<code>Quit</code>	Flash Lite Player の再生を停止して終了します。
<code>ResetSoftKeys</code>	ソフトキーを元の設定にリセットします。
<code>SetInputTextType</code>	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。
<code>SetQuality</code>	アニメーションのレンダリング画質を設定します。
<code>SetSoftKeys</code>	アクセスして再マッピングできる場合には、デバイスの左と右のソフトキーを再マッピングします。
<code>StartVibrate</code>	携帯端末のバイブレータ機能を作動させます。
<code>StopVibrate</code>	バイブレータが動作している場合は、それを停止します。
<code>Unescape</code>	安全にネットワーク上に送信できるようにエンコードされた任意のストリングを、通常のエンコードされていない形式にデコードします。

# 機能

この項では、Macromedia Flash Lite 1.1で認識可能なプラットフォーム機能と変数について説明します。各項目は、先行アンダースコアを無視してアルファベット順に示します。

## `_capCompoundSound`

### 使用できるバージョン

Flash Lite 1.1

### シンタックス

`_capCompoundSound`

### 説明

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。処理できる場合は、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。

例として、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 Player 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。

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");  
}
```

## `_capEmail`

### 使用できるバージョン

Flash Lite 1.1

### シンタックス

`_capEmail`

## 説明

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");
}
```

# \_capLoadData

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

`_capLoadData`

## 説明

数値変数。ホストアプリケーションが `loadMovie()`、`loadMovieNum()`、`loadVariables()`、および `loadVariablesNum()` 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、`iCanLoad` 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");
}
```

# \_capMFi

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_capMFi

## 説明

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。再生できる場合はこの変数が定義され、1 に設定されます。できない場合は定義されません。

## 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) {
    // イベントサウンドを起動するボタンがあるフレームに movieclip ボタンを送ります
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

# \_capMIDI

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_capMIDI

## 説明

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、canMidi を 1 に設定します。

```
canMIDI = _capMIDI;

if (canMIDI == 1) {
    // イベントサウンドを起動するボタンがあるフレームに movieclip ボタンを送ります
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

# \_capMMS

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_capMMS

## 説明

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) {
        // MMS を送信します
        myMessage = "mms:4156095555?body=sample mms message";
    } else {
        // SMS を送信します
        myMessage = "sms:4156095555?body=sample sms message";
    }
    getURL(myMessage);
}
```

# **\_capMP3**

## **使用できるバージョン**

Flash Lite 1.1

## **シンタックス**

`_capMP3`

## **説明**

数値変数。デバイスがサウンドデータを MP3 (MPEG Audio Layer 3) オーディオ形式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。

## **例**

次の例では、デバイスが MP3 サウンドデータを再生できる場合、`canMP3` を 1 に設定します。

```
canMP3 = _capMP3;
if (canMP3 == 1) {
    tellTarget("soundClip") {
        gotoAndPlay(2);
    }
}
```

# **\_capSMAF**

## **使用できるバージョン**

Flash Lite 1.1

## **シンタックス**

`_capSMAF`

## **説明**

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) {
    // イベントサウンドを起動するボタンがあるフレームに movieclip ボタンを送ります
    tellTarget("buttons") {
        gotoAndPlay(2);
    }
}
```

# \_capSMS

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_capSMS

## 説明

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。再生できる場合はこの変数が定義され、1 に設定されます。できない場合は定義されません。

## 例

次の例では、Flash Lite 1.1 の場合は canSMS 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 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) {
        // SMS を送信します
        myMessage = "sms:4156095555?body=sample sms message";
        getURL(myMessage);
    }
}
```

# \_capStreamSound

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_capStreamSound

## 説明

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) {
        // このボタンで movieclip のストリーミングサウンドを再生します
        tellTarget("music") {
            gotoAndPlay(2);
        }
    }
}
```

# \_cap4WayKeyAS

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

\_cap4WayKeyAS

## 説明

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。

この変数の値が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 joystick 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.";
}
```

# \$version

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
$version
```

## 説明

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 Player 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;
```

# fscommand()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
status = fscommand("Launch", "application-path, arg1, arg2,..., argn")
```

## パラメータ

"Launch" コマンド指定子。Launch コマンドは、fscommand() 関数を使用して実行する唯一のコマンドです。

"application-path, arg1, arg2,..., argn" コンマで区切った開始されるアプリケーションの名前とそのパラメータ。

## 説明

関数。SWF ファイルが Flash Lite、または携帯端末やデバイスのオペレーティングシステムなどのホスト環境と通信できるようにします。

## 関連項目

[fscommand2\(\)](#)

# Launch

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
status = fscommand("Launch", "application-path, arg1, arg2,..., argn")
```

## パラメータ

"Launch" コマンド指定子。Flash Lite では、fscommand() 関数を使用するのは Launch コマンドを実行するときだけです。

"application-path, arg1, arg2,..., argn" コンマで区切った開始されるアプリケーションの名前とそのパラメータ。

## 説明

`fscommand()` 関数を使用して実行されるコマンドです。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。



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、Flash Lite Player がスタンドア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 Player 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。

## 例

次の例では、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");
}
```

## 関連項目

[fscommand2\(\)](#)

# fscommand2()

## 使用できるバージョン

Flash Lite 1.1

## シンタックス

```
returnValue = fscommand2(command [, expression1 ... expressionN])
```

## パラメータ

*command* ホストアプリケーションに任意の用途で渡される文字列、または Flash Lite に渡されるコマンド。

*parameter1...parameterN* コマンド区切りの文字列で、*command* で指定したコマンドへパラメータとして渡されます。

## 説明

関数。SWF ファイルが Flash Lite、または携帯端末やデバイスのオペレーティングシステムなどのホスト環境と通信できるようにします。fscommand2() が返す値はコマンドによって異なります。

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、任意数の引数を取れます。
- fscommand2() はすぐに実行されますが、fscommand() は処理中のフレームの最後に実行されます。
- fscommand2() 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。コマンドやパラメータとしてこの関数に渡す文字列および式については、この項の表で説明しています。

各テーブルには次の 3 つの列があります。

- [ コマンド ] 列には、コマンドを識別する文字列リテラルパラメータを示します。
- [ パラメータ ] 列では、追加パラメータがある場合に、渡す値の種類を説明します。
- [ 戻り値 ] 列では、予測される戻り値について説明します。

## 例

fscommand2() 関数を使用して実行する特定のコマンドの例については、この項の後半を参照してください。

## 関連項目

[fscommand\(\)](#)

# Escape

## 使用できるバージョン

Flash Lite 1.1

## 説明

任意のストリングをネットワーク転送に安全な形式にエンコードします。英数字以外の文字を 16 進数のエスケープシーケンスに置き換えます (マルチバイト文字の場合は、%XX、または %XX%XX)。

コマンド	パラメータ	戻り値
"Escape"	<i>original</i> URL として使用できる形式にエンコードするストリング <i>encoded</i> エンコードされたストリング 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、"Encoded_String")。	0: 失敗 1: 成功

## 例

次の例は、サンプルストリングをエンコードされた形式に変換する方法を示しています。

```
original_string = "Hello, how are you?";  
status = fscommand2("escape", original_string, "encoded_string");  
trace(encoded_string); // 出力 : Hello%2C%20how%20are%20you%3F
```

## 関連項目

[Unescape](#)

# FullScreen

## 使用できるバージョン

Flash Lite 1.1

## 説明

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。全画面表示にすることも、それ以下のサイズにすることもできます。

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にのみサポートされます。Flash Lite Player 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
"FullScreen"	<i>size</i> 定義済みの変数、または true (全画面表示) か false (全画面表示より小さい) のいずれかの定数ストリング値。これ以外の値は false 値とみなされます。	-1: サポートされない 0: サポートされる

### 例

次の例では、表示領域を全画面表示に設定します。戻り値が 0 以外の場合は、再生ヘッドが `smallScreenMode` というフレームへ送られます。

```
status = fscommand2("FullScreen", true);
if(status != 0) {
    gotoAndPlay("smallScreenMode");
}
```

## GetBatteryLevel

### 使用できるバージョン

Flash Lite 1.1

### 説明

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。

コマンド	パラメータ	戻り値
"GetBatteryLevel"	なし	-1: サポートされていない その他の数値: 現在のバッテリーの容量

### 例

次の例では、変数 `battLevel` を現在のバッテリーの容量に設定します。

```
battLevel = fscommand2("GetBatteryLevel");
```

### 関連項目

[GetMaxBatteryLevel](#)

# GetDateDay

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在の日付を返します。返される値は数値です (先頭に 0 は付きません)。有効な日は 1 ~ 31 です。

コマンド	パラメータ	戻り値
"GetDateDay"	なし	-1: サポートされていない 1 ~ 31: 月内の日付

## 例

次の例では、日付情報を収集して完全な日付のストリングを構成します。

```
today = fscommand2("GetDateDay");  
weekday = fscommand2("GetDateWeekday");  
thisMonth = fscommand2("GetDateMonth");  
thisYear = fscommand2("GetDateYear");  
when = weekday add ", " add ThisMonth add " " add today add ", " add thisYear;
```

## 関連項目

[GetDateMonth](#)、[GetDateWeekday](#)、[GetDateYear](#)

# GetDateMonth

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在の日付の月を数値で返します (先頭に 0 は付きません)。

コマンド	パラメータ	戻り値
"GetDateMonth"	なし	-1: サポートされていない 1 ~ 12: 現在の月を表す数値

## 例

次の例では、日付情報を収集して完全な日付のストリングを構成します。

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add thisMonth add " " add today add ", " add thisYear;
```

## 関連項目

[GetDateDay](#)、[GetDateWeekday](#)、[GetDateYear](#)

# GetDateWeekday

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在の日付の曜日に相当する数字を数値としてを返します。

コマンド	パラメータ	戻り値
"GetDateWeekday"	なし	-1: サポートされていない 0: 日曜 1: 月曜 2: 火曜 3: 水曜 4: 木曜 5: 金曜 6: 土曜

## 例

次の例では、日付情報を収集して完全な日付のストリングを構成します。

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add thisMonth add " " add today add ", " add thisYear;
```

## 関連項目

[GetDateDay](#)、[GetDateMonth](#)、[GetDateYear](#)

# GetDateYear

現在の日付の年を表す 4 桁の数値を返します。

コマンド	パラメータ	戻り値
"GetDateYear"	なし	-1: サポートされていない 0 ~ 9999: 現在の年

## 使用できるバージョン

Flash Lite 1.1

## 例

次の例では、日付情報を収集して完全な日付のストリングを構成します。

```
today = fscommand2("GetDateDay");  
weekday = fscommand2("GetDateWeekday");  
thisMonth = fscommand2("GetDateMonth");  
thisYear = fscommand2("GetDateYear");  
when = weekday add ", " add thisMonth add " " add today add ", " add thisYear;
```

## 関連項目

[GetDateDay](#)、[GetDateMonth](#)、[GetDateWeekday](#)

# 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"	<i>device</i> デバイスの ID を受け取るストリング。変数名または変数名を含むストリング値を使用できます。	-1: サポートされていない 0: サポートされている

## 使用できるバージョン

Flash Lite 1.1

## 例

次のコードの例では、デバイスの ID を `statusdevice` 変数に代入して、汎用デバイス名でテキストフィールドを更新します。

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 506i 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

```
statusdevice = fscommand2("GetDevice", "devicename");
switch(devicename) {
    case "D506i":
        /:myText += "device: Mitsubishi 506i" add newline;
        break;
    case "DFOMA1":
        /:myText += "device: Mitsubishi FOMA1" add newline;
        break;
    case "F506i":
        /:myText += "device: Fujitsu 506i" add newline;
        break;
    case "FFOMA1":
        /:myText += "device: Fujitsu FOMA1" add newline;
        break;
    case "N506i":
        /:myText += "device: NEC 506i" add newline;
        break;
    case "NFOMA1":
        /:myText += "device: NEC FOMA1" add newline;
        break;
    case "Nokia 3650":
        /:myText += "device: Nokia 3650" add newline;
        break;
    case "P506i":
        /:myText += "device: Panasonic 506i" add newline;
```

```

        break;
    case "PFOMA1":
        /:myText += "device: Panasonic FOMA1" add newline;
        break;
    case "SH506i":
        /:myText += "device: Sharp 506i" add newline;
        break;
    case "SHFOMA1":
        /:myText += "device: Sharp FOMA1" add newline;
        break;
    case "S0506i":
        /:myText += "device: Sony 506i" add newline;
        break;
}

```

## 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"	<i>id</i> 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: サポートされていない 0: サポートされている

### 使用できるバージョン

Flash Lite 1.1

### 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

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: サポートされていない 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

---

## 使用できるバージョン

Flash Lite 1.1

## 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");
```

## 関連項目

[GetTotalPlayerMemory](#)

# GetLanguage

## 使用できるバージョン

Flash Lite 1.1

現在デバイスで使用されている言語を識別するパラメータを設定します。言語は、名前で渡される変数に文字列として返されます。

コマンド	パラメータ	戻り値
"GetLanguage"	<p><i>language</i> 言語コードを受け取る文字列。これは変数の名前、または変数の名前が含まれる文字列値です。戻り値は次のいずれかです。</p> <p>cs: チェコ語 da: デンマーク語 de: ドイツ語 en-UK: 英語 ( 英国 ) または英語 ( 国際 ) en-US: 英語 ( 米国 ) es: スペイン語 fi: フィンランド語 fr: フランス語 hu: ハンガリー語 it: イタリア語 ja: 日本語 ko: 韓国語 nl: オランダ語 no: ノルウェー語 pl: ポーランド語 pt: ポルトガル語 ru: ロシア語 sv: スウェーデン語 tr: トルコ語 xu: 判別できない言語 zh-CN: 簡体中国語 zh-TW: 繁体中国語</p>	<p>-1: サポートされていない 0: サポートされている</p>

※

日本の携帯端末が英語を表示するように設定されている場合は、en\_US が *language* として返されます。

## 例

次の例では、言語コードを変数 language に代入し、Flash Lite Player が認識した言語でテキストフィールドを更新します。

```
statusLanguage = fscommand2("GetLanguage", "language");
switch(language) {
    case "cs":
        /:myText += "language is Czech" add newline;
        break;
    case "da":
        /:myText += "language is Danish" add newline;
        break;
    case "de":
        /:myText += "language is German" add newline;
        break;
    case "en-UK":
        /:myText += "language is UK" add newline;
        break;
    case "en-US":
        /:myText += "language is US" add newline;
        break;
    case "es":
        /:myText += "language is Spanish" add newline;
        break;
    case "fi":
        /:myText += "language is Finnish" add newline;
        break;
    case "fr":
        /:myText += "language is French" add newline;
        break;
    case "hu":
        /:myText += "language is Hungarian" add newline;
        break;
    case "it":
        /:myText += "language is Italian" add newline;
        break;
    case "jp":
        /:myText += "language is Japanese" add newline;
        break;
    case "ko":
        /:myText += "language is Korean" add newline;
        break;
    case "nl":
        /:myText += "language is Dutch" add newline;
        break;
    case "no":
        /:myText += "language is Norwegian" add newline;
        break;
    case "pl":
        /:myText += "language is Polish" add newline;
```

```

        break;
    case "pt":
        /:myText += "language is Portuguese" add newline;
        break;
    case "ru":
        /:myText += "language is Russian" add newline;
        break;
    case "sv":
        /:myText += "language is Swedish" add newline;
        break;
    case "tr":
        /:myText += "language is Turkish" add newline;
        break;
    case "xu":
        /:myText += "language is indeterminable" add newline;
        break;
    case "zh-CN":
        /:myText += "language is simplified Chinese" add newline;
        break;
    case "zh-TW":
        /:myText += "language is traditional Chinese" add newline;
        break;
}

```

## GetLocaleLongDate

### 使用できるバージョン

Flash Lite 1.1

### 説明

パラメータを、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を表す文字列 (長い形) に設定します。

コマンド	パラメータ	戻り値
"GetLocaleLongDate"	<p><i>longdate</i> 現在の日付の値を長い形式で受け取る文字列変数 ("October 16, 2004", "16 October 2004" など)。</p> <p>変数名または変数名を含む文字列値を使用できます。</p> <p><i>longdate</i> に返される値は、複数文字からなる可変長の文字列です。実際のフォーマットはデバイスとロケールによって異なります。</p>	<p>-1: サポートされない</p> <p>0: サポートされる</p>

## 例

次の例では、longDate 変数を使用して現在の日付を長い形式で返します。さらに、status の値も設定して、処理を実行できたかどうかを報告します。

```
status = fscommand2("GetLocaleLongDate", "longdate");  
trace (longdate);           // 出力 : Tuesday, June 14, 2005
```

## 関連項目

[GetLocaleShortDate](#)、[GetLocaleTime](#)

# GetLocaleShortDate

## 使用できるバージョン

Flash Lite 1.1

## 説明

パラメータを、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を表す文字列 (短い形式) に設定します。

コマンド	パラメータ	戻り値
"GetLocaleShortDate"	<i>shortdate</i> 現在の日付を短い形式で受け取る文字列変数 ("10/16/2004"、"16-10-2004" など)。変数名または変数名を含む文字列値を使用できます。 <i>shortdate</i> に返される値は、複数文字からなる可変長の文字列です。実際のフォーマットはデバイスとロケールによって異なります。	-1: サポートされない 0: サポートされる

## 例

次の例では、短い形式の現在日付を変数 shortDate に代入します。さらに、status の値も設定して、処理を実行できたかどうかを報告します。

```
status = fscommand2("GetLocaleShortDate", "shortdate");  
trace (shortdate);       // 出力 : 06/14/05
```

## 関連項目

[GetLocaleLongDate](#)、[GetLocaleTime](#)

# GetLocaleTime

## 使用できるバージョン

Flash Lite 1.1

## 説明

パラメータを、現在定義されている地域設定に従って書式設定された、現在時刻を表すストリングに設定します。

コマンド	パラメータ	戻り値
"GetLocaleTime"	<i>time</i> 現在時刻の値を受け取るストリング変数 ("6:10:44 PM"、-1: サポートされない "18:10:44" など)。 変数名または変数名を含むストリング値を使用できます。 0: サポート <i>time</i> に返される値は、複数文字からなる可変長のストリングで される す。実際のフォーマットはデバイスとロケールによって異なり ます。	

## 例

次の例では、現在のローカルタイムを *time* 変数に代入します。さらに、*status* の値も設定して、処理を実行できたかどうかを報告します。

```
status = fscommand2("GetLocaleTime", "time");  
trace(time); // 出力 : 14:30:21
```

## 関連項目

[GetLocaleLongDate](#)、[GetLocaleShortDate](#)

# GetMaxBatteryLevel

## 使用できるバージョン

Flash Lite 1.1

## 説明

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。

コマンド	パラメータ	戻り値
"GetMaxBatteryLevel"	なし	-1: サポートされていない 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

## 使用できるバージョン

Flash Lite 1.1

## 説明

最大信号強さのレベルを返します。返される値は、正の数値です。

コマンド	パラメータ	戻り値
"GetMaxSignalLevel"	なし	-1: サポートされていない 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

## 使用できるバージョン

Flash Lite 1.1

## 説明

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: サポートされていない 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); // 出力 : 80
```

## 関連項目

[GetVolumeLevel](#)

# GetNetworkConnectStatus

## 使用できるバージョン

Flash Lite 1.1

## 説明

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。

コマンド	パラメータ	戻り値
"GetNetworkConnectStatus"	なし	-1: サポートされていない 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) {
    case -1 :
        /:myText += "connectstatus not supported" add newline;
        break;
    case 0 :
        /:myText += "connectstatus shows active connection" add newline;
        break;
    case 1 :
        /:myText += "connectstatus shows attempting connection" add newline;
        break;
    case 2 :
        /:myText += "connectstatus shows no connection" add newline;
        break;
    case 3 :
        /:myText += "connectstatus shows suspended connection" add newline;
        break;
    case 4 :
        /:myText += "connectstatus shows indeterminable state" add newline;
        break;
}
```

# GetNetworkName

## 使用できるバージョン

Flash Lite 1.1

## 説明

パラメータを現在のネットワーク名に設定します。

コマンド	パラメータ	戻り値
"GetNetworkName"	<i>networkName</i> ネットワーク名を表すストリング。変数名または変数名を含むストリング値を使用できます。 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、 <i>networkname</i> 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: サポートされていない 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変数 `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

## 使用できるバージョン

Flash Lite 1.1

## 説明

最新の HTTP 要求のステータスを示す値を返します。

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:
        /:myText += "requeststatus not supported" add newline;
        break;
    case 0:
        /:myText += "connection to server has been made" add newline;
        break;
    case 1:
        /:myText += "connection is being established" add newline;
        break;
    case 2:
        /:myText += "pending request, contacting network" add newline;
        break;
```

```

case 3:
    /:myText += "pending request, resolving domain" add newline;
    break;
case 4:
    /:myText += "failed, network error" add newline;
    break;
case 5:
    /:myText += "failed, couldn't reach server" add newline;
    break;
case 6:
    /:myText += "HTTP error" add newline;
    break;
case 7:
    /:myText += "DNS failure" add newline;
    break;
case 8:
    /:myText += "request has been fulfilled" add newline;
    break;
case 9:
    /:myText += "request timedout" add newline;
    break;
case 10:
    /:myText += "no HTTP request has been made" add newline;
    break;
}

```

## GetNetworkStatus

### 使用できるバージョン

Flash Lite 1.1

### 説明

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: コマンドがサポートされていない 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:
    /:myText += "network status not supported" add newline;
    break;
  case 0:
    /:myText += "no network registered" add newline;
    break;
  case 1:
    /:myText += "on home network" add newline;
    break;
  case 2:
    /:myText += "on extended home network" add newline;
    break;
  case 3:
    /:myText += "roaming" add newline;
    break;
}
```

# GetPlatform

## 使用できるバージョン

Flash Lite 1.1

## 説明

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"	<i>platform</i> プラットフォームの ID を受け取る文字列。変数名または変数名を含む文字列値を使用できます。	-1: サポートされない 0: サポートされる

## 例

次のコードの例では、デバイスのIDを `statusplatform` 変数に代入して、汎用プラットフォーム名でテキストフィールドを更新します。

`myPlatform` の結果のサンプルと、それが意味するデバイスのクラスを次に示します。

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

```
statusplatform = fscommand2("GetPlatform", "platform");
switch(platform){
  case "506i":
    /:myText += "platform: 506i" add newline;
    break;
  case "FOMA1":
    /:myText += "platform: FOMA1" add newline;
    break;
  case "Symbian6.1-Series60v1":
    /:myText += "platform: Symbian6.1, Series 60 version 1 phone" add newline;
    break;
  case "Symbian7.0":
    /:myText += "platform: Symbian 7.0" add newline;
    break;
}
```

# GetPowerSource

## 使用できるバージョン

Flash Lite 1.1

## 説明

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。

コマンド	パラメータ	戻り値
"GetPowerSource"	なし	-1: サポートされていない 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

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在の信号の強さを数値で返します。

コマンド	パラメータ	戻り値
"GetSignalLevel"	なし	-1: サポートされていない 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");
```

# GetTimeHours

## 使用できるバージョン

Flash Lite 1.1

## 説明

24 時間表記で現在時刻の時を返します。返される値は数値です (先頭に 0 は付きません)。

コマンド	パラメータ	戻り値
"GetTimeHours"	なし	-1: サポートされていない 0 ~ 23: 現在時刻の時

## 例

次の例では、hour 変数を現在時刻の時間または -1 に設定します。

```
hour = fscommand2("GetTimeHours");  
trace (hour); // 出力 : 14
```

## 関連項目

[GetTimeMinutes](#)、[GetTimeSeconds](#)、[GetTimeZoneOffset](#)

# GetTimeMinutes

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在時刻の分を返します。返される値は数値です (先頭に 0 は付きません)。

コマンド	パラメータ	戻り値
"GetTimeMinutes"	なし	-1: サポートされていない 0 ~ 59: 現在時刻の分

## 例

次の例では、minutes 変数を現在時刻の分または -1 に設定します。

```
minutes = fscommand2("GetTimeMinutes");  
trace (minutes);           // 出力 : 38
```

## 関連項目

[GetTimeHours](#)、[GetTimeSeconds](#)、[GetTimeZoneOffset](#)

# GetTimeSeconds

## 使用できるバージョン

Flash Lite 1.1

## 説明

現在時刻の秒を返します。返される値は数値です (先頭に 0 は付きません)。

コマンド	パラメータ	戻り値
"GetTimeSeconds"	なし	-1: サポートされていない 0 ~ 59: 現在時刻の秒

## 例

次の例では、seconds 変数を現在時刻の秒または -1 に設定します。

```
seconds = fscommand2("GetTimeSeconds");  
trace (seconds);           // 出力 : 41
```

## 関連項目

[GetTimeHours](#)、[GetTimeMinutes](#)、[GetTimeZoneOffset](#)

# GetTimeZoneOffset

## 使用できるバージョン

Flash Lite 1.1

## 説明

パラメータを、ローカルタイムゾーンと世界時 (UTC) の差を表す数値 (分単位) に設定します。

コマンド	パラメータ	戻り値
"GetTimeZoneOffset"	<i>timezoneOffset</i> ローカルタイムゾーンと UTC の差を表す分。これは変数の名前、または変数の名前が含まれる文字列値です。次のように、正または負の数値が返されます。 540: 日本標準時 -420: 太平洋標準時 (夏時間)	-1: サポートされていない 0: サポートされている

## 例

次の例では、UTC からの時差の分数を変数 `timezoneoffset` に代入して `status` を 0 に設定するか、または `status` を -1 に設定します。

```
status = fscommand2("GetTimeZoneOffset", "timezoneoffset");  
trace (timezoneoffset);// 出力 : 300
```

## 関連項目

[GetTimeHours](#)、[GetTimeMinutes](#)、[GetTimeSeconds](#)

# GetTotalPlayerMemory

## 使用できるバージョン

Flash Lite 1.1

## 説明

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: サポートされていない 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");
```

## 関連項目

[GetFreePlayerMemory](#)

# GetVolumeLevel

## 使用できるバージョン

Flash Lite 1.1

## 説明

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: サポートされていない 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);           // 出力 : 50
```

## 関連項目

[GetVolumeLevel](#)

# Quit

## 使用できるバージョン

Flash Lite 1.1

## 説明

Flash Lite Player 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にのみサポートされます。Flash Lite Player 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: サポートされていない

#### 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

### 使用できるバージョン

Flash Lite 1.1

### 説明

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 がスタンドア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ます。Flash Lite Player 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: サポートされていない 0: サポートされている

#### 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");
```

### 関連項目

[SetSoftKeys](#)

# SetInputTextType

## 使用できるバージョン

Flash Lite 1.1

## 説明

入力テキストフィールドを開くモードを指定します。

コマンド	パラメータ	戻り値
"SetInputTextType"	<i>variableName</i> 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 文字列値です。 <i>type</i> Numeric、Alpha、Alphanumeric、Latin、 NonLatin、または NoRestriction のいずれかの値	0: 失敗 1: 成功

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に要求することで、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。

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。

指定モード	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)	

×  
#

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");
```

# SetQuality

## 使用できるバージョン

Flash Lite 1.1

## 説明

アニメーションのレンダリング品質を設定します。

コマンド	パラメータ	戻り値
"SetQuality"	<i>quality</i> レンダリング品質は "high"、 "medium"、または "low" 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、レンダリング品質を low に設定します。

```
status = fscommand2("SetQuality", "low");
```

# SetSoftKeys

## 使用できるバージョン

Flash Lite 1.1

## 説明

デバイスの左ソフトキーと右ソフトキーを再マッピングします (アクセスおよび再マッピングが可能な場合)。

このコマンドの実行後、左のキーを押すと PageUp keypress イベントが発生し、右のキーを押すと PageDown keypress イベントが発生します。該当するキーが押されたときに、PageUp および PageDown の各 keypress イベントに関連付けられている ActionScript が実行されます。

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 Player 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
"SetSoftKeys"	<i>left</i> 左のソフトキーに表示するテキスト <i>right</i> 右の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、左のソフトキーに「Previous」、右のソフトキーに「Next」と表示します。

```
status = fscommand2("SetSoftKeys", "Previous", "Next");
```

## 関連項目

[ResetSoftKeys](#)

# StartVibrate

## 使用できるバージョン

Flash Lite 1.1

## 説明

携帯端末のバイブレータ機能を開始します。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 Player 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"	<i>time_on</i> 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) <i>time_off</i> 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) <i>repeat</i> この動作を繰り返す回数 (最大 3 回)	-1: サポートされていない 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。

```
status = fscommand2("StartVibrate", 2500, 1000, 2);
```

## 関連項目

[StopVibrate](#)

# StopVibrate

## 使用できるバージョン

Flash Lite 1.1

## 説明

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: サポートされていない 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");
```

## 関連項目

[StartVibrate](#)

# Unescape

## 使用できるバージョン

Flash Lite 1.1

## 説明

ネットワーク転送を安全に行うためにエンコードされた任意の文字列をデコードして、通常のエンコードされていない形にします。16進形式の文字 ( パーセント文字 (%) の後に 2 桁の 16 進値が付いた形式 ) は、すべてデコードされた形式に変換されます。

コマンド	パラメータ	戻り値
"Unescape"	<i>original</i> URL として使用できる形式から通常の形式にデコードする文字列 <i>decoded</i> デコードされた文字列 ( このパラメータは変数の名前、または変数の名前が含まれる文字列値です )	0: 失敗 1: 成功

## 例

次の例は、エンコードされたストリングのエンコードを示しています。

```
encoded_string = "Hello%2C%20how%20are%20you%3F";  
status = fscommand2("unescape", encoded_string, "normal_string");  
trace (normal_string);    // 出力 : Hello, how are you?
```

## 関連項目

[Escape](#)

# 索引

## 記号

!(否定(NOT))演算子 86  
\$version 変数 115  
%(剰余)演算子 88  
%=(剰余を代入)演算子 89  
&&(論理積(AND))演算子 86  
-(減算)演算子 102  
--(デクリメント)演算子 82  
,(カンマ)演算子 79  
◇(数値不等価)演算子 93  
""(ストリング区切り記号)演算子 96  
.(ドット)演算子 84  
(否定(NOT))演算子 86  
>(より大きい)演算子 92  
>(より大きいか等しい)演算子 93  
<(より小さい-数値)演算子 94  
<(より小さいか等しい-数値)演算子 95  
(論理積(AND))演算子 86  
(論理和(OR))演算子 87  
\*(乗算)演算子 90  
\*=(乗算後代入)演算子 89  
+(数値加算)演算子 91  
++(インクリメント)演算子 84  
+=(加算後代入)演算子 76  
-=(減算後代入)演算子 103  
/(除算)演算子 82  
/(スラッシュ-ルートタイムライン)プロパティ 48  
/\*(ブロックコメント)演算子 78  
//(コメント)演算子 80  
/=(除算)演算子 83  
=(代入)演算子 78  
==(数値等価)演算子 91  
?(条件演算子)演算子 81  
\_alpha 変数 49  
\_cap4WayKeyAS 変数 114  
\_capCompoundSound 変数 108  
\_capEmail 変数 108

\_capLoadData 変数 109  
\_capMFi 変数 110  
\_capMIDI 変数 110  
\_capMMS 変数 111  
\_capSMAF 変数 112  
\_capSMS 変数 113  
\_capStreamSound 変数 114  
\_currentframe プロパティ 50  
\_focusrect プロパティ 50  
\_framesloaded プロパティ 51  
\_height プロパティ 51  
\_highquality プロパティ 52  
\_level プロパティ 52  
\_name プロパティ 54  
\_rotation プロパティ 54  
\_scroll プロパティ 55  
\_target プロパティ 56  
\_totalframes プロパティ 56  
\_visible プロパティ 57  
\_width プロパティ 57  
\_x プロパティ 58  
\_xscale プロパティ 58  
\_y プロパティ 59  
\_yscale プロパティ 60  
||(論理和(OR))演算子 87

## A

add(ストリング連結)演算子 76  
AND 演算子 86  
and 演算子 77

## B

break ステートメント 62

## C

call 13  
case ステートメント 63  
chr() 関数 14  
continue ステートメント 64

## D

do..while ステートメント 65  
duplicateMovieClip() 関数 14

## E

else if ステートメント 67  
else ステートメント 66  
eq ( ストリング等価 ) 演算子 97  
eval() 関数 16

## F

for ステートメント 68  
for ループ 68  
fscommand() コマンド 116

## G

ge ( より大きいか等しい - ストリング ) 演算子 98  
getProperty() 関数 17  
getTimer() 関数 17  
getURL() 関数 18  
gotoAndPlay() 関数 20  
gotoAndStop() 関数 21  
gt ( より大きい - ストリング ) 演算子 97

## I

if ステートメント 69  
ifFrameLoaded() 関数 22  
int() 関数 23

## L

le ( より小さいか等しい - ストリング ) 演算子 101  
length() 関数 23  
loadMovie() 関数 24  
loadMovieNum() 関数 25  
loadVariables() 関数 27  
loadVariablesNum() 関数 28  
lt ( より小さい - ストリング ) 演算子 100

## M

maxscroll プロパティ 53  
mbchr() 関数 29  
mbsubstring() 関数 31  
MFI サウンド 110  
MIDI サウンド 110  
MMS メッセージング 111

## N

ne ( ストリング不等価 ) 演算子 99  
nextFrame() 関数 32  
nextScene() 関数 32  
NOT 演算子 86  
Number() 関数 33

## O

on() 関数 34  
OR 演算子 87  
ord() 関数 35

## P

play() 関数 35  
prevFrame() 関数 36  
prevScene() 関数 36

## R

random() 関数 37  
removeMovieClip() 関数 38

## S

scroll プロパティ 55  
set() 関数 38  
setProperty() 関数 39  
stop() 関数 40  
stopAllSounds() 関数 40  
String() 関数 41  
substring() 関数 42  
switch ステートメント 69

## T

tellTarget() 関数 42  
toggleHighQuality() 関数 43  
trace() 関数 44

## U

unloadMovie() 関数 45  
unloadMovieNum() 関数 46

## W

while ステートメント 71  
while ループ 65

## い

インクリメント演算子 84

## え

演算子

(否定 (NOT)) 86  
(論理積 (AND)) 86  
and 77  
インクリメント 84  
加算後代入 76  
カンマ 79  
減算後代入 103  
コメント 80  
条件ステートメント 81  
乗算 90  
剰余 88  
剰余の代入 89  
除算 82  
除算後代入 83  
数値加算 91  
数値等価 91  
数値不等価 93  
文字列区切り記号 96  
文字列等価 97  
文字列不等価 99  
文字列連結 76  
文字列、より大きいか等しい 98  
文字列、より小さいか等しい 101  
代入 78  
ドット 84  
ブロックコメント 78

より大きい 92  
より大きい - 文字列 97  
より大きいか等しい 93  
より小さい - 数値 94  
より小さい - 文字列 100  
より小さいか等しい - 数値 95  
論理和 (OR) 87

## か

関数

chr() 14  
duplicateMovieClip() 14  
eval() 16  
fscommand() 116  
getProperty() 17  
getTimer() 17  
getURL() 18  
gotoAndPlay() 20  
gotoAndStop() 21  
iffFrameLoaded() 22  
int() 23  
length() 23  
loadMovie() 24  
loadMovieNum() 25  
loadVariables() 27  
loadVariablesNum() 28  
mbchr() 29  
mbsubstring() 31  
nextFrame() 32  
nextScene() 32  
Number() 33  
on() 34  
ord() 35  
play() 35  
prevFrame() 36  
prevScene() 36  
random() 37  
removeMovieClip() 38  
set() 38  
setProperty() 39  
stop() 40  
stopAllSounds() 40  
String() 41  
substring() 42  
tellTarget() 42  
toggleHighQuality() 43  
trace() 44  
unloadMovie() 45  
unloadMovieNum() 46

カンマ演算子 79

## け

減算後代入演算子 103

## こ

コメント

1行 80

block 78

## さ

サウンドの変数 108、110、112、114

## し

条件 69

条件演算子 81

乗算 90

剰余演算子 88

剰余の代入 89

除算 82

除算後代入演算子 83

## す

数値加算 91

ステートメント

(否定 (NOT)) 86

break 62

case 63

continue 64

do..while 65

else 66

else if 67

for 68

if 69

switch 69

while 71

文字列区切り記号演算子 96

文字列等価演算子 97

文字列、より大きい演算子 97

文字列、より大きいか等しい 98

文字列、より小さいか等しい 101

## た

代入演算子 78

## て

電子メール機能の変数 108

## と

ドット演算子 84

## ふ

不等価演算子 93

ブロックコメント演算子 78

プロパティ

\_alpha 49

\_currentframe 50

\_focusrect 50

\_framesloaded 51

\_height 51

\_highquality 52

\_level 52

\_name 54

\_rotation 54

\_scroll 55

\_target 56

\_visible 57

\_width 57

\_x 58

\_xscale 58

\_y 59

\_yscale 60

maxscroll 53

scroll 55

スラッシュ 48

## へ

変数

\$version 115

\_alpha 49

\_cap4WayKeyAS 114

\_capCompoundSound 108

\_capEmail 108

\_capLoadData 109

\_capMFi 110

\_capMIDI 110

\_capMMS 111

\_capSMAF 112

\_capSMS 113

\_capStreamSound 114

Flash Lite のバージョン番号 115

データをロードする機能 109

電子メール機能 108

方向キーによる移動 114

変数、サウンド

\_capCompoundSound 108

\_capMFi 110

\_capMIDI 110

\_capSMAF 112

\_capStreamSound 114

変数、メッセージング

\_capMMS 111

\_capSMS 113

## め

メッセージングの変数 111、113

## よ

より大きい演算子 92

より大きいか等しい演算子 93

より小さい演算子 94

より小さいか等しい演算子 95

## る

ルートタイムライン識別子 48

## れ

連結 76

