macromedia®

FLASH™5

ActionScript

macromedia

Obsah

ÚVOD

Začátek

ActionScript je Flash skriptovací jazyk, který používáme ke kontrole objektů ve Flash animacích, pro vytváření navigačních a interaktivních prvků a při vytváření vysoce interaktivních animací a Web aplikací.

Co je nového ve Flash 5 ActionScriptu

Flash 5 ActionScript nabízí nové zajímavé vlastnosti ke tvorbě působivých, interaktivních Web stránek plných důmyslných her, formulářů, přehledů a chatů v reálném čase.

Flash 5 ActionScript má mnoho nových rysů a pravidel syntaxe, které ho dělají velice podobným programovacímu jazyku JavaScript. Tento manuál vysvětluje základní programovací koncepty jako jsou funkce, proměnné, výrazy, operátory, podmínky a smyčky. Kapitola 7 tohoto manuálu „ActionScript Slovník“ obsahuje detailní popis každého prvku ActionScriptu.

Cílem tohoto manuálu není dotýkat se programování všeobecně, na to existuje mnoho dostupných zdrojů, které poskytují daleko více informací o obecných konceptech programování a jazyku JavaScript.

European Computers Manufacturers Association (ECMA) (Asociace Evropských Výrobců Počítačů) napsala dokument nazvaný ECMA-262, který byl odvozen z JavaScriptu, aby sloužil jako mezinárodní standard pro jazyk JavaScript. ActiosScript je založený na specifikaci ECMA-262, která je dostupná na http://www.ecma.ch.

NetscapeDevEdge Online má JavaScript Developer
(http://developer.netscape.com/tech/javascript/index.html),
který obsahuje dokumentaci a články užitečné pro pochopení ActionScript.
Nejhodnotnějším zdrojem je Core JavaScript Guide, umístěný
na http://developer.netscape.com/docs/manuals/js/core/jsguide/index.html.

Rozdíly mezi ActionScript a JavaScript

Nemusíte vědět, jak se používá JavaScript a přesto se můžete učit se ActionScript. Jestliže znáte JavaScript, potom vám bude ActionScript připadat známý. Některé rozdíly mezi ActionScript a JavaScript:

- ActionScript nepodporuje specifické objekty browseru, jako Document, Window a Anchor.
- ActionScript nepodporuje kompletně všechny předdefinované objekty JavaScriptu.
- ActionScript podporuje konstrukce syntaxe, které nejsou povoleny v JavaScript (například akce `tellTarget a ifFrameLoaded a slash syntax`).
- ActionSctipt nepodporuje některé syntaxe konstrukce JavaScriptu, jako `switch`, `continue`, `try`, `catch`, `throw`, and `statement` popisky.
- ActionScript nepodporuje JavaScript konstruktor `Function`.
- V ActionScriptu akce `eval` může provést pouze odkazy na proměnnou.
- V JavaScriptu, `toString undefined` je `undefined`. Ve Flash 5, pro kompatibilitu s Flash 4, `toString undefined` je " ".
- V JavaScriptu, ohodnocování `undefined` je numerický kontext jehož výsledkem je NaN. Ve Flash 5, pro kompatibilitu s Flash 4, je výsledkem ohodnocení undefined 0.
- ActionScript nepodporuje Unicode; podporuje ISO-8859-1 a nastavení znaku Shift-JIS.

Editování textu

Skripty můžete vkládat přímo do panelu Actions v Expert Mode. Také můžete zvolit prvky z pop-up menu nebo z Toolbox jako jste to dělali ve Flash 4.

Dot syntax

Dot syntax, můžete použít pro získání a nastavení vlastností a metod objektu, včetně instancí filmových klipů a proměnných. Můžete použít dot syntax namísto slash syntax používaném ve Flash 4. Slash syntax již není preferován, ale je stále podporován ve Flash Přehrávači.

Typy dat

Flash 5 ActionScript podporuje následující typy dat: string, number, boolean, object a movie clip. Několik typů dat umožňuje použít různé typy informací v ActionScriptu. Například můžete vytvářet arrays a associative arrays.

Lokální proměnné

Můžete deklarovat lokální proměnné, kterým končí platnost na konci seznamu akcí nebo volání funkce. Toto umožňuje řídit paměť a znovu použít jména proměnných. Proměnné Flash 4 byly všechny permanentní - dokonce dočasné proměnné jako počítadla smyček zůstávaly v animaci dokud neskončila.

Funkce definované uživatelem

Můžete definovat funkce s parametry, které dávají hodnoty. Toto umožňuje znovu použít bloky kódů ve vašich skriptech. Ve Flash 4 jste mohli znovu použít kód použitím akce `call`, ale nemohli jste propustit parametry nebo obdržet hodnoty.

Předdefinované objekty

Pro dosažení a manipulaci s určitými typy informací můžete používat předdefinované objekty. Několik předdefinovaných objektů:

- Objekt Math vystupuje jako plný doplněk zabudování matematických konstant a funkcí, jako je E (Eulerova konstanta), `cos` (Cosinus) a `atan` (Arctangent).
- Objekt Dat umožňuje získat informaci o datumu a čase, v jakémkoliv systému, který provozuje Flash Přehrávač.
- Objekt Sound umožňuje přidat a kontrolovat zvuky v animaci, během přehrávání. Například můžete upravit hlasitost (`setVolume`) nebo rovnováhu (`setPan`).
- Objekt Mouse umožňuje schovat standardní kursor tak, že můžete používat upravený kursor.
- Objekt MovieClip umožňuje kontrolovat filmové klipy bez použití obalové akce, jako `tellTarget`. Můžete zavolat metodu jako `play`, `loadMovie` nebo `duplicateMovieClip` ze jména instance použitím dot syntax (například, `myMovieClip.play ()`.

Akce klipů

Pro stanovení akcí přímo do instancí filmového klipu na Scéně, můžete použít akci `onClipEvent`. Akce `onClipEvent` má události jako `load`, `enterFrame`, `mouseMove` a `data`, které vám umožňují vytvořit nové druhy pokročilé interaktivity.

Nové akce

Můžete použít nové akce jako `do..while` a `for` k vytváření komplexních smyček. Další nové akce jsou implementovány jako metody MovieClip objektu; například `getBounds`, `attachMovie`, `hitTest`, `swapDepths a globalToLocal`.

Smart klipy
Smart klipy mají vnitřní scénáře, které vy nebo jiný vývojář můžete změnit bez použití panelu Akce. Můžete propustit hodnoty do Smart klipu pomocí parametrů klipu, které můžete definovat v Knihovně.

Debugger
Debugger umožňuje prohlédnout si a změnit proměnnou a hodnoty vlastnosti v animaci, která hraje v režimu Test-movie, samostatném Přehrávači nebo Webovém prohlížeči. Toto umožňuje jednoduše najít problémy v ActionScriptu.

XML podpora
Předdefinovaný XML objekt umožňuje konvertovat ActionScript na XML dokumenty a použít je do aplikací serveru. Také můžete použít XML objekt pro natažení XML dokumentů do Flash animace a jejich interpretaci. Předdefinovaný XML Socket objekt umožňuje vytvářet plynulé spojení se serverem k propouštění XML dat pro aplikace v reálném čase.

Použití Flash nápovědy pro akce
Flash 5 obsahuje kontextově-citlivou pomoc pro každou akci dosažitelnou v panelu Akce. Když vytváříte scénáře, můžete dostat informace o akcích, které používáte.

Jak získat nápovědu o akcích:
1. V panelu Actions zvolte akci ze seznamu.

2. Klikněte na tlačítko Help na vrcholu panelu.

V prohlížeči se objeví téma vztahující se k akci.

KAPITOLA 1

Pochopení ActionScriptu
ActionScript je Flash skriptovací jazyk, který přidává animacím interaktivitu. Animaci můžete nastavit tak, že uživatelovy události, jako je kliknutí na tlačítko a stisknutí klávesy, spustí skripty, které říkají animaci, jakou akci má provést. Například můžete napsat skript, který říká Flashi, aby natáhl různé animace do Flash Přehrávače v závislosti na tom, jaké navigační tlačítko uživatel zvolí.

Přemýšlejte o ActionScriptu jako o nástroji, který umožňuje vytváření animací, chovajících se přesně podle vašich přání. Abyste mohli začít skriptovat, nemusíte chápat všechna možná použití nástrojů, ale jestliže máte jasný cíl, můžete začít tvořit skripty s jednoduchými akcemi. Můžete zabudovávat nové prvky jazyka tak, jak se je naučíte.

Tato kapitola vás uvádí do ActionScriptu jako **objektově orientovaného skriptovacího jazyka** a poskytuje přehled termínů ActionScriptu. Také uvádí příkladové skripty, abyste mohli tvorit narocnejsi práce.

Tato kapitola vás také uvádí do Actions panel, kde můžete vytvořit skripty volbou prvků ActionSctripu nebo vložením textu do Script window.

Skriptování v ActionScriptu
Můžete začít psaní jednoduchých skriptů bez velikých znalostí ActionScriptu. Vše co potřebujete je záměr; potom je to pouze záležitost výběru správných akcí. Nejlepší způsob jak se naučit používat ActionScript, je vytvoření vlastního skriptu. Následující kroky připojují skript ke tlačítku, které mění viditelnost filmového klipu.

Jak změnit viditelnost filmového klipu:
1. Zvolte Window > Common Libraries > Buttons a potom zvolte Window > Common Libraries > Movie Clips. Umístěte tlačítko a filmový klip na Scénu.

2. Vyberte instanci filmového klipu na Scéně a zvolte Window > Panels > Instance Properties.

3. Do pole Name vložte **testMC**.

4. Zvolte tlačítko na Scéně a zvolte Window > Actions, pro otevření panelu Akce.

5. V panelu Object Actions klikněte na kategorii Akce, pro její otevření.

6. Dvakrát klikněte na akci **`setProperty`**, pro její přidání do seznamu Akce.

7. Z pop-up menu Property zvolte **_visible**.

8. Jako parametr Target vložte **testMC**.

9. Jako parametr Value vložte **0**.

Kód by měl vypadat takto:

```
on (release){
setProperty ("testMC",_visible,false);
}
```

10. Zvolte Control > Test Movie a klikněte na tlačítko, abyste viděli, jak movie klip zmizí.

ActionScript je objektově orientovaný skriptovací jazyk. To znamená, že když se objeví určitá událost, tak akce kontrolují objekty. V tomto skriptu je **událostí uvolnění myši, objekt je instance filmového klipu `testMC` a akce je `setProperty`**. Když uživatel klikne na tlačítko na scéně, událost **`release`** spustí skript, který nastaví vlastnost objektu **`MC`**, **`_visible`** na **`false`** (nepravda) a způsobí tak, že objekt se stane neviditelný.

Panel Akcí můžete použít jako vodítko pro nastavování jednoduchých skriptů. Pro použití plného výkonu ActionScript je důležité pochopit jak jazyk funguje: koncepty, prvky a pravidla, která jazyk používá pro organizovaní informací a vytváření interaktivních animací.

Tato část vysvětluje průběh ActionScriptu, základní koncepty objektově orientovaného skriptování, Flash objekty a tok skriptu. Také popisuje, kde jsou ve Flash animaci skripty umístěny.

O plánovaní a odstraňování závad skriptů

Když píšete skripty pro celé animace, počet a různost skriptů může být značně rozsáhlá. Rozhodování o tom, které akce použít, jak efektivně strukturovat skript a kam mají být skripty umístěny, vyžaduje pečlivé plánování a testování, obzvláště při růstu složitosti a náročnosti celé animace.

Ještě než začnete psát skripty, je třeba si vytyčit a formulovat cíl, kterého chceme animací dosáhnout. Je to stejně důležité jako sepsání a postavení osnovy celého projektu. Začněte náčrtkem toho, co se má v animaci dít, podle tohoto příkladu:

- Chci vytvořit celou svou stránku s použitím Flashe.

- Návštěvníci stránky budou tázáni na jméno, které bude znovu používáno v odkazech na stránkách.

- Stránka bude mít tažitelný navigační sloupec s tlačítky, které jsou spojeny s každou částí webu.

- Když je kliknuto na tlačítko, nová část se rozetmí do středu Scény.

- Jedna scéna bude mít kontaktní formulář s již vyplněným uživatelovým jménem.

Když víte, co chcete, můžete začít vytvářet potřebné objekty, a začít psát skripty pro ovládání těchto objektů.

Vypracování skriptů, které pracují přesně způsobem jakým jste si to naplánovali trvá nějakou dobu a také dlouho trvá testování a odstraňování závad. Nejlepší způsob je začít jednoduše a **velmi často** svou **práci kontrolovat a testovat**. Pokud se dopracujete k jedné fungující části, je nejlepší ji uložit – Save As (například - mojeAnimace.fla), a dát se do práce na další části. Takový postup vám pomůže efektivně identifikovat závady a ujistí vás, že váš ActionScript je solidní a můžete se pustit do složitějších věcí.

O objektově orientovaném skriptování

V objektově orientovaném skriptování organizujete informace jejich aranžováním do skupin nazvaných **třídy** (classes). Pro použití ve skriptech, můžete vytvořit několik instancí tříd, nazývaných **objekty** (objects). Můžete použít buď předdefinované třídy ActionScriptu a nebo vytvořit své vlastní.

Při vytvoření třídy, definujete všechny **vlastnosti** (properties) a **metody** (methods) každého vytvořeného objektu. Například, osoba má vlastnosti jako pohlaví výška a barva vlasů a metody jako řeč, chůze. V tomto příkladě je „osoba“ třída a každá individuální osoba je objekt nebo instance této třídy.

Objekty v ActionScript mohou obsahovat data, nebo mohou být graficky reprezentovány na Scéně jako filmové klipy. Všechny filmové klipy jsou instance předdefinovaných tříd MovieClip. Každá instance filmového klipu obsahuje všechny vlastnosti (například **`_height`**, **`_rotation`**, **`_totalframes`** a všechny metody (například **`gotoAndPlay`**, **`loadMovie`**, **`startDrag`**) MovieClip třídy.

Abyste definovali třídu, vytvořte speciální funkci nazvanou **konstrukční funkce** (constructor function); předdefinované třídy mají konstrukční funkce, které už jsou definované. Například, jestliže chcete informaci o cyklistovi ve vašem klipu, můžete vytvořit konstrukční funkci **Cyklista**, s vlastnostmi **cas** a **vzdalenost** a metodou **urcit**, která vám řekne, jak rychle cyklista jede:

```
function Cyklista (c, v) {
      this.cas = c;
      this.vzdalenost = v;
}
function Rychlost () {
      return this.cas / this.vzdalenost;
}
Cyklista.prototype.urcit = Rychlost;
```

Potom můžete vytvořit kopie - to znamená instance - třídy. Následující kód vytváří instance objektu **Cyklista** nazvané **petr** a **pavel**.

```
petr = new Cyklista (30, 5);
pavel = new Cyklista (40, 5);
```

Instance také mohou komunikovat mezi sebou. Pro objekt **Cyklista** můžete vytvořit metodu nazvanou **strcit**, která umožní jednomu cyklistovi strčit do druhého. (Instance **petr** může zavolat svou metodu **strcit**, jestliže se **pavel** dostane příliš blízko). Pro předání informace do metody použijete parametry (argumenty): například metoda **strcit** by mohla mít parametry **koho** a **jakDaleko**. V tomto příkladě **petr** strká **pavla** o 10 pixel:

```
petr.strcit (pavel, 10);
```

V objektově orientovaném skriptování mohou třídy obdržet vlastnosti a metody od sebe navzájem v určitém pořadí; toto je nazýváno **dědičnost** (inheritance). Dědičnost můžete použít pro rozšíření nebo redefinovaní vlastností a metod třídy. Třída, která dědí od jiné třídy je nazývána **subtřída** (subclass). Třída, která definuje vlastnosti a metody jiné třídě je nazývána **supertřída** (superclass). Třída může být jak subtřída, tak supertřída.

O objektu MovieClip

Předdefinované třídy ActionScriptu jsou nazývány **objekty** (objects). Každý objekt vám umožňuje dosáhnout určitého typu informace. Například objekt **Date** má metody (například **getFullYear, getMonth**), které vám umožňují číst informaci ze systémového času. Objekt **Sound** má metody (například **setVolume, setPan**), které vám umožňují kontrolovat zvuk v animaci. Objekt **MovieClip** má metody umožňující kontrolovat instance filmového klipu (například **play, stop** a **getURL**) a získat a nastavit informaci o jejich vlastnostech (například **_alpha,_framesloaded,_visible**).

MovieClipy jsou nejdůležitějšími objekty Flash animaci, protože mají Časovou osu), která běží nezávisle na sobě. Například, jestliže hlavní Časová osa má pouze jeden snímek a filmový klip v tomto snímku má deset snímků, tak časová osa v klipu bude neustále přehrávána. Toto umožňuje instancím hrát jako autonomní objekty, které mohou navzájem komunikovat.

Každá instance filmového klipu má unikátní jméno, takže ji můžete cílovat akcí. Například, můžete mít několik instancí na Scéně (například **levyKlip** a **pravyKlip**) a chcete, aby v daném čase hrála pouze jedna. Abyste připojili akci, která říká jedné určité instanci, aby hrála, potřebujete použít její jméno. V následujícím příkladu je jméno instance filmového klipu **levyKlip**.

```
levyKlip.play ();
```

Jména instancí vám také umožňují duplikovat, odstraňovat a táhnout filmové klipy, zatímco animace hraje. Následující příklad duplikuje instanci **kartovaPolozka**, pro vyplnění nákupní karty číslem nakupované položky:

```
onClipEvent (load) {
     do {
        duplicateMovieClip ("kartovaPolozka", "kartovaPolozka" + i, i);
          i = i + 1;
     } while (i <= cisloNakupPolozky);
}
```

Filmové klipy mají vlastnosti, jejichž hodnoty můžete s ActionsScriptem dynamicky nastavit a znovu získat. Změna a přečtení těchto vlastností může změnit vzhled a identitu filmového klipu a je klíčem pro vytvoření interaktivity. Například, následující scénář používá akci **setProperty** pro nastavení průhlednosti (nastavení alfa) instance **navigacniSloupec** na 10:

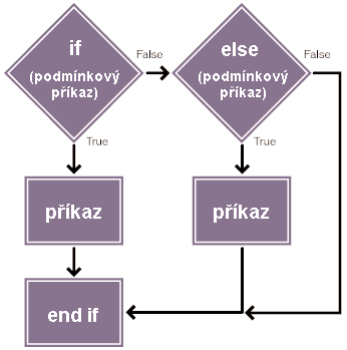
```
setProperty ("navigacniSloupec", _alpha, 10);
```

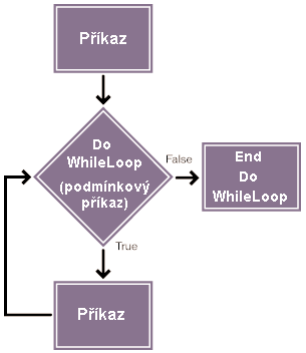
Více informací o dalších typech objektů viz „Použití předdefinovaných objektů“.

Jak skripty pracují

ActionScript vyplývá z logického toku. Flash vykonává příkazy ActionScriptu tak, že začne prvním příkazem a pokračuje dalšími skripty podle pořadí, dokud nedosáhne posledního příkazu nebo příkazu, který ActionScript posílá někam jinam.

Jedny z akcí, které posílají ActionScript někam jinam než na další příkaz jsou příkazy **if**, smyčky **do...while** a akce **return**.





Příkaz **if** je nazýván také jako podmínkový příkaz nebo „logická větev“, protože kontroluje tok skriptu založeného na ohodnocení určité podmínky. Například následující kód kontroluje, zda je hodnota proměnné **pocet** menší nebo rovna 10. Jestliže obdrží **true** (pravda) (například hodnota **pocet** je 5), je nastavena proměnná **varovani** a zobrazí její hodnotu do textového pole:

```
if ( pocet <= 10 ) {
      varovani = "Počet je menší než nebo roven 10" ;
}
```


Také můžete přidat příkazy **else** pro vytvoření složitějších podmínkových výrazů. Na následujícím příkladě vidíme, že pokud podmínka obdrží (má hodnotu) **true** (pravda) (například hodnota **pocet** je 3), běží příkaz mezi prvním nastavením složených závorek a proměnná **varovani** je nastavena na druhém řádku. Pokud ale podmínka obdrží **false** (nepravda) (například hodnota **pocet** je 30), první blok kódu je přeskočen a za příkazem **if** mezi složenými závorkami běží příkaz **else**:

```
if ( pocet <= 10 ) {
      varovani = "Počet je menší než nebo roven 10" ;
} else {
      varovani = "Počet je větší než 10" ;
}
```

Více informací viz „Použití příkazů “**if**“.

Smyčka opakuje akci několikrát, nebo dokud není splněna určitá podmínka. V následujícím příkladě je filmový klip pětkrát duplikován:

```
i = 0 ;
do {
   duplicateMovieClip („mujKlip", „novyKlip" + i, i) ;
   noveJmeno = eval („novyKlip" + i) ;
   setProperty (noveJmeno, _x, getProperty („mujKlip", _x) + (i * 5)) ;
   i = i + 1;
} while (i <= 5) ;
```

Detailní informace viz „Opakování akce“

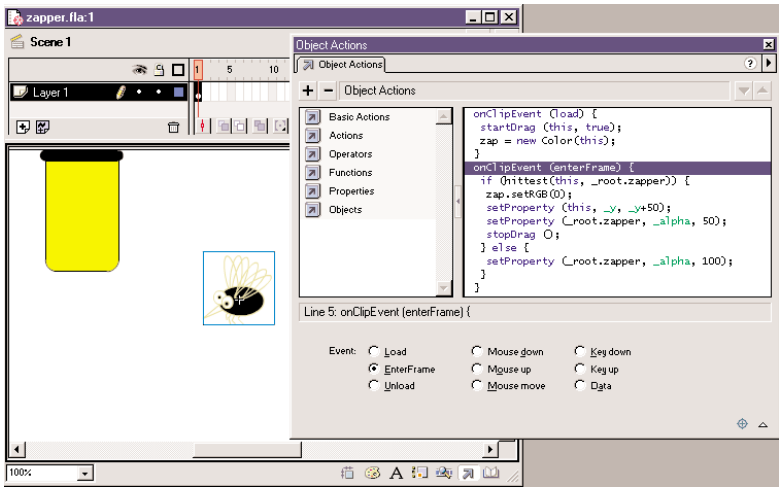
Kontrola ActionScriptu za běhu

Při psaní skriptů používáte Action panel, který umožňuje připojit skripty ke snímku na hlavní Časové ose, nebo ke snímku na Časové ose jakéhokoliv filmového klipu. Samozřejmě i k tlačítkům a filmovým klipům umístěným přímo na scéně.

Flash vykonává akce v různém čase v závislosti na tom, k čemu jsou připojeny:

- Akce připojené ke snímku, jsou vykonány jakmile je dosaženo tohoto snímku na Časové ose.
- Akce připojené k tlačítku, jsou vloženy do akcí ovladače **on**.
- Akce připojené k filmovému klipu jsou vloženy do akcí ovladače **onClipEvent**.

onClipEvent a akce **on** jsou nazvané ovladače, protože „ovládají“ nebo řídí událost. (Událost je příhoda jako pohyb myši, stisknutí klávesy nebo natahování movie klipu). Movie klip a tlačítkové akce jsou vykonány při události specifikované ovladačem. Jestliže připojíte k objektu více než jeden ovladač, akce budou vykonávány při různých událostech. Více informací viz Kapitola 3, „Vytváření Interaktivity v ActionScripu“.



Terminologie ActionScriptu

Jako kterýkoliv skriptovací jazyk, také ActionScript používá specifickou terminologii vzhledem ke specifickým pravidlům syntaxe. Následující seznam poskytuje úvod do důležitých termínů ActionScriptu v abecedním pořadí. Tyto řídící termíny a syntaxe, jsou probírány detailněji v Kapitole 2, „Psaní Skriptů v ActionScripu.“

Actions (Akce) jsou příkazy, které instruují animaci, aby něco během přehrávání vykonala. Například **gotoAndStop** posílá hrací hlavu na určitý snímek nebo popisek. V této knize jsou termíny akce a příkaz zaměnitelné.

Arguments (Argumenty), také nazývané parametry, jsou schránky umožňující předávání a definici hodnot do funkcí. Například, následující funkce, nazvaná **welcome**, používá dvě hodnoty obdržené v argumentech firstName a hobby:

```
function welcome (firstName, hobby) {
        welcomeText = "Ahoj," +firstName+ "Vidím, že tě baví" +hobby ;
}
```


Classes (Třídy) jsou typy dat vytvářené, pro definování nového typu objektu. Třídy objektu definujeme konstrukční funkcí.

Constants (Konstanty) jsou prvky, které se nemění. Například konstanta **TAB** má vždy stejný význam. Konstanty jsou užitečné pro porovnávání hodnot.

Constructors (Konstruktory) jsou funkce, které používáte pro definování vlastností a metod třídy. Například následující kód vytváří novou třídu vytvořením konstrukční funkce nazvané **Circle**:

```
function Circle(x, y, radius) {
    this.x = x ;
    this.y = y ;
    this.radius = radius ;
}
```

Data types (Typy dat) jsou sady hodnot a operací, které na nich mohou být provedeny. String, number, (Booleovské) hodnoty **true** a **false**, object a movie clip jsou typy ActionScript dat. Více detailů o těchto prvcích jazyka viz „O typech dat“.

Events (Události) jsou akce, které se objevují při přehrávání animace. Některé příklady událostí: animace se natahuje a hrací hlava přijede na určitý snímek, uživatel klikne na tlačítko nebo movie klip, nebo uživatel použije klávesnici atd.

Expressions (Výrazy) jsou části příkazů produkující nějakou hodnotu. Například **2+2** je výraz.

Functions (Funkce) jsou bloky znovu použitelného kódu, který může předávat argumenty (parametry) a vracet hodnotu. Například, funkce **getProperty**, předává jméno vlastnosti a jméno instance movie klipu a vrací hodnotu vlastnosti. Funkce **getVersion** udává verzi Flash Přehrávače právě přehrávané animace.

Handlers (Ovladače) jsou speciální akce, které „ovládají“ nebo řídí události jako **mouseDown** nebo **load**. Například **on(onMouseEvent)** a **onClipEvent** jsou ActionScript ovladače.

Identifiers (Identifikátory) jsou jména používaná pro určení proměnné, vlastnosti, objektu, funkce nebo metody. První znak musí být písmeno, dolní pomlčka (_) nebo znak dolaru ($). Každý následující znak musí být písmeno, číslo, dolní pomlčka (_) nebo znak dolaru ($). Například **firstName** je jméno proměnné.

Instances (Instance) jsou objekty patřící do určité třídy. Každá instance třídy obsahuje všechny vlastnosti a metody této třídy. Všechny movie klipy jsou instancemi s vlastnostmi (například **_alpha** a **_visibility**) a metodami (například **gotoAndPlay** a **getURL**) MovieClip třídy.

Instance names (Jména instancí) jsou jedinečná jména umožňující cílování instancí movie klipů ve skriptech. Například symbol v Knihovně by mohl být nazván **`counter`** a dvě instance tohoto symbolu v movie klipu by mohly mít jména instancí **`scorePlayer1`** a **`scorePlayer2`**. Následující kód nastavuje proměnnou nazvanou **`score`** uvnitř každé instance klipu použitím jmen instancí:

```
_root.scorePlayer1.score += 1
_root.scorePlayer2.score -= 1
```

Keywords (Klíčová slova) jsou rezervovaná slova, která mají zvláštní význam. Například **`var`** je klíčové slovo pro určení lokálních proměnných.

Methods (Metody) jsou funkce připojené k objektu. Metodou objektu se stává pouze připojená funkce. Například v následujícím kódu, **`clear`** se stává metodou objektu **`controller`**:

```
function Reset ( ) {
    x_pos = 0 ;
    y_pos = 0 ;
}
controller.clear = Reset ;
contoller.clear ( ) ;
```

Objects (Objekty) jsou sbírky vlastností; každý objekt má své vlastní jméno a hodnotu. Objekty umožňují dosáhnout určitého typu informace. Například předdefinovaný objekt **`Date`** poskytuje informaci ze systémového času.

Operators (Operátory) jsou termíny, které přepočítávají novou hodnotu nebo více hodnot. Například připojení operátoru (+) spojuje dvě nebo více hodnot dohromady pro vytvoření nové hodnoty.

Target paths (Cílové cesty) jsou hierarchické adresy jmen movie klip instancí, proměnných a objektů v animaci. Instanci klipu můžete pojmenovat v panelu Instance. Hlavní Časová osa bude mít vždy jméno **`_root`**. Cílová cesta se používá k přímé akci v movie klipu nebo k získání nebo nastavení hodnoty proměnné. Například následující příkaz je cílovou cestou k proměnné **`volume`** uvnitř movie klipu **`stereoControl`**:

```
_root.stereoControl.volume
```

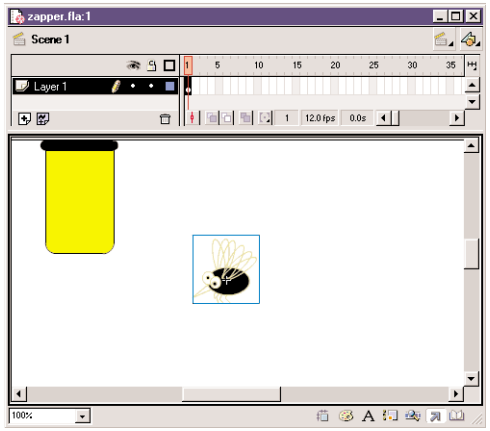
Properties (Vlastnosti) jsou atributy definující objekt. Například **`_visibility`** je vlastnost movie klipů, která určuje, zda je movie klip viditelný nebo je schován.

Variables (Proměnné) jsou identifikátory, které obsahují hodnoty jakéhokoliv typu dat. Proměnné mohou být vytvořeny, změněny a updatovány. Hodnoty, které jsou do nich ukládány mohou být znovu získávány pro použití v dalších skriptech. V následujícím příkladu jsou proměnnými, identifikátory na levé straně od znaménka rovná se:

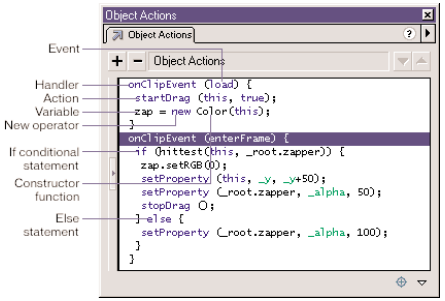
```
x = 5;
jmeno ="Pavel";
zakaznik.adresa ="Nova 25";
c = new Color(mcinstanceJmeno);
```

Rozbor ukázkového skriptu

V této ukázkové animaci, kde uživatel táhne bug (štěnici) do bug zapper (likvidátoru štěnic), štěnice se otočí, zčerná a spadne do likvidátoru štěnic. Animace je jeden snímek dlouhá a obsahuje dva objekty, movie klip instanci štěnice a movie klip instanci likvidátoru štěnic. Každý movie klip obsahuje také pouze jeden snímek.



V animaci je pouze jeden skript; je připojen do instance **`bug`** jak je uvedeno dole v panelu Object Actions:



Oba objekty musí být movie klipy, abyste jim mohli přiřadit jméno instance v panelu Instance a manipulovat s nimi pomocí ActionScriptu. Jméno instance štěnice je **bug** a jméno instance likvidátoru štěnic je **zapper**. Ve skriptu je na **bug** odkazováno jako na **this**, protože skript je připojen k **bug** a vyhrazené slovo **this** odkazuje na objekt, který ho volá.

Jsou zde dva ovladače **onClipEvent** se dvěma různými událostmi: **load** a **enterFrame**. Akce v příkazu **onClipEvent**(**load**) se vykonají pouze jednou a to, když se movie stáhne. Akce v příkazu **onClipEvent**(**enterFrame**) se vykonají pokaždé, když hrací hlava přijede do snímku. Dokonce v jedno-snímkovém movie hrací hlava vstupuje opakovaně do tohoto snímku a skript je opakovaně vykonáván. Následující akce se objeví uvnitř každého ovladače **onClipEvent**:
onClipEvend(load) Akce **startDrag** učiní movie klip bug (štěnice) tažitelný. Instance objektu Color je vytvořena pomocí operátoru **new** a Color konstrukční funkce **Color**, a připojena k proměnné **zap**:

```
onClipEvent ( load ) {
       startDrag ( this, true ) ;
       zap = new Color (this) ;
}
```

onClipEvent(enterFrame) Podmínkový výraz **if** ohodnotí a zkontroluje akcí **hitTest**, zda se instance bug (**this**) dotýká instance bug zapper (**_root.zapper**). Jsou zde dva možné výstupy ohodnocení, **true** nebo **false**:

```
onClipEvent ( enterFrame ) {
      if ( this.hitTest ( _root.zapper )) {
           zap.setRGB (0) ;
           setProperty ( _target, _y, _y + 50 ) ;
           setProperty ( _root.zapper, _alpha, 50 ) ;
           stopDrag ( ) ;
      } else {
           setProperty ( _root.zapper, _alpha, 100 ) ;
      }
}
```

Jestliže akce **hitTest** je **true**, objekt **zap** vytvořený událostí **load** je použit pro nastavení barvy bug (štěnice) na černou. Vlastnost štěnice (**_y**) je nastavena na ni samotnou plus **50**, takže bug (štěnice) spadne. Průhlednost likvidátoru štěnic (zapper)(**_alpha**) je nastavena na **50**, takže je matný. Akce **stopDrag** zastaví tažení štěnice.

Jestliže akce **hitTest** je **false**, běží v akci následující příkaz **else**, ve kterém je hodnota **_alpha** likvidátoru štěnic (zapper) nastavena na **100**. Toto způsobí, že likvidátor štěnic se bude blýskat, protože hodnota **_alpha** jde z původního stavu (100) na přepnutý stav (50) a zpět na původní stav. Akce **hitTest** je **false** a příkaz else se vykonají poté, co bug (štěnice) byla přepnuta a spadla.

Chcete-li animaci vidět hrát, podívejte se na Flash Help.

Používání panelu Akce

Panel Akcí umožňuje vytvářet a editovat akce pro objekty nebo snímeky, za použití dvou různých režimů editace. Akce můžete zvolit již předepsané ze seznamu Toolbox list, tažením a puštěním je vkládat, nebo používat tlačítka pro jejich rušení a změnu. V Normal Mode můžete psát akce použitím polí parametrů (argumentů), které vás nasměrují ke správným argumentům. V Expert Mode můžete psát a editovat akce přímo do textového boxu, podobně jako psaní skriptů v textovém editoru.

Zobrazení panelu Akce:
Zvolte Window >Actions.

Zvolení instance tlačítka nebo movie klipu učiní panel Akce aktivní. Název panelu Akce se změní na Object Actions, jestliže je zvoleno tlačítko nebo movie klip a na panel Frame Actions, jestliže je zvolen snímek.

Zvolení editovacího režimu:

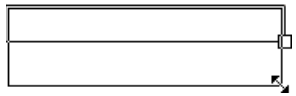
1. Se zobrazeným panelem Akce klikněte na šipku v pravém horním rohu panelu pro zobrazení pop-up menu.

2. Zvolte Normal Mode nebo Expert Mode z pop-up menu.

Každý skript má svůj vlastní režim. Skriptovat můžete například jednu instanci tlačítka v Normálním Režimu a jinou v Expert Režimu. Přepínání mezi zvoleným tlačítkem přepíná také režim panelu.

Normální Režim

V Normálním Režimu můžete vytvářet akce zvolením akcí ze seznamu nazvaného Toolbox list, umístěného na levé straně panelu,. Tento seznam obsahuje Basic Actions, Actions, Operators, Functions, Properties a Objects. Kategorie Basic Actions obsahuje nejjednodušší Flash akce a je dostupná pouze v Normálním Režimu. Zvolené akce jsou zapsány na pravé straně panelu v seznamu Akcí. Pořadí příkazů akcí můžete zrušit změnit, nebo akce přidat; také můžete vložit parametry (argumenty) pro akce do pole parametrů na spodní části panelu.
V Normálním Režimu, můžete používat pro zrušení nebo změnu pořadí příkazů v seznamu, také kontrolu v panelu Akce. Kontroly jsou zvláště užitečně pro ovládání akcí snímku a tlačítkových akcí, které se zkládají z několika příkazů.



Zvolení akce:

1. Klikněte na kategorii Akce v Toolbox listu, pro zobrazení akcí v této kategorii.

2. Dvakrát klikněte na akci nebo ji přetáhněte do okna Skript.

Použití polí Parametrů:

1. Pro zobrazení polí, klikněte na tlačítko Parameters v pravém spodním rohu panelu Akce.

2. Zvolte akci a vložte nové hodnoty do polí Parameters, pro změnu parametrů existujících akcí.

Vložení cílové cesty movie klipu:

1. Klikněte na tlačítko Target Path v pravém spodním rohu panelu Akcí, pro zobrazení dialog boxu Target Path.

2. Zvolte movie klip ze zobrazeného seznamu.

Pohyb příkazu nahoru a dolů v seznamu:

1. Zvolte příkaz v seznamu Akcí.

2. Klikněte na směrové šipky v pravém horním rohu - **Up** nebo **Down**.

Zrušení akce:

1. Zvolte příkaz v seznamu Akcí.

2. Klikněte na tlačítko Delete (–).

Změna parametrů existujících akcí:

1. Zvolte příkaz v seznamu Akcí.

2. Vložte nové hodnoty do polí Parameters.

Pro změnu velikosti Toolbox listu, nebo seznamu Akcí, udělejte jedno z následujících:

- Přetáhněte vertikální rozdělovací sloupec, který se objeví mezi Toolbox listem a seznamem Akcí.

- Dvakrát klikněte na rozdělovací sloupec pro schování Toolbox listu; dvakrát klikněte opět na sloupec pro jeho znovuzobrazení.

- Klikněte na šipku směřující vlevo, nebo vpravo, umístěnou ve středu rozdělovacího sloupce, pro schování, nebo znovuzobrazení Toolbox listu.

Jestliže je Toolbox list schován, můžete stále používat jeho položky použitím Add (+) tlačítko vlevo nahoře v panelu Akcí.

Expert Režim

V Expert Režimu vytváříte akce vkládáním ActionScriptu do textového boxu na pravé straně panelu, nebo zvolením akcí z Toolbox listu. Akce můžete editovat, vkládat jejich parametry, nebo zrušit akce přímo v textovém boxu, podobně jako vytváříte skripty v textovém editoru.

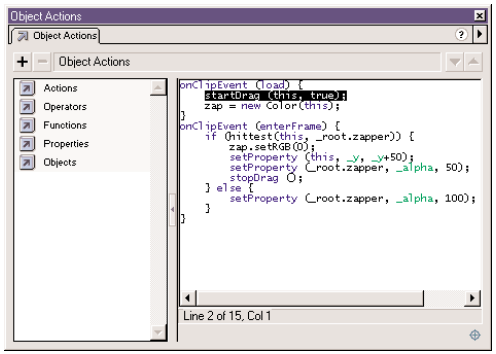
Expert Režim umožňuje pokročilým uživatelům ActionScriptu editovat své skripty pomocí textového editoru, jako by tvořili JavaScript nebo VBScript. Expert Režim se liší od Normálního Režimu v těchto parametrech:

- Zvolení položky použitím tlačítka Add (+) nebo v Toolbox listu vloží položku do oblasti textové editace.

- Neobjeví se žádné pole parametrů.

- V tlačítkovém panelu funguje pouze tlačítko Add (+).

- Šipky Up a Down na přesouvání akcí zůstávají neaktivní.



Přepínání mezi editovacími režimy

Změna editovacích režimů při tvorbě skriptu, může změnit jeho formátování. Z tohoto důvodu je nejlepší používat jeden editovací režim na každý skript.

Když přepínáte z Normálního na Expert Režim, odsazení a formátování je zachováno. Ačkoliv skripty z Normálního režimu můžete konvertovat i s chybami do Expert Režimu, nemůžete skripty s chybami exportovat.

Přepínání z Expert Režimu do Normálního Režimu je trochu komplikované:

- Když přepínáte na Normální Režim, Flash přeformátuje skript a rozebere jakékoliv bílé místo a odsazení, které jste přidali.
- Jestliže přepínáte na Normální Režim a potom zpátky na Expert Režim, Flash přeformátuje skript podle jeho vzhledu v Normálním Režimu.
- Skripty z Expert Režimu obsahující chyby nemohou být exportovány nebo konvertovány na Normální Režim; jestliže se skript snažíte konvertovat, obdržíte hlášení o chybě.

Přepínání editovacích režimů:

Zvolte Normální Režim nebo Expert Režim z pop-up menu vpravo nahoře panelu Akce. Kontrolní značka indikuje zvolený režim.

Nastavení preferencí editovacího režimu:

1. Zvolte Edit > Preferences.
2. Zvolte General tab
3. V části Actions Panel zvolte Normální Režim nebo Expert Režim z pop-up menu.

Používání externího editoru

Ačkoliv Expert Režim panelu Akcí vám dává dostatek možností při editaci ActionScriptu, můžete si také zvolit editaci skriptu mimo Flash. Pro přidání skriptů do Flashe, napsaných v externím editoru, použijte akci **include**.

Například následující příkaz importuje soubor se skriptem:

```
#include "externisoubor.txt"
```

Text souboru se skriptem nahradí akci **include**. Při exportu movie, musí být ale textový soubor přítomen.

Přidání scénářů napsaných v externím editoru do scénáře uvnitř Flash:

1. Přetáhněte akci **include** z Toolbox listu do okna Skript.
2. Vložte cestu k externímu souboru do boxu Path.

Cesta by měla být relativní k FLA souboru. Například jestliže **myMovie.fla** a **externisoubor.txt** jsou ve stejné složce, cesta by byla **externisoubor.txt**. Jestliže je **externisoubor.txt** v podsložce nazvané **Skripty**, cesta by byla **skripty/externisoubor.txt**.

Výběr variant v panelu Akcí

Panel Akcí vám umožňuje pracovat se scénáři (skripty) různými způsoby. Můžete změnit velikost fontu v okně Skript, můžete importovat textový soubor obsahující ActionScript do panelu Akcí a exportovat akce jako textový soubor, můžete hledat a nahrazovat text ve scénáři a používat zvýraznění syntaxe, pro snadnější čtení skriptů a snadnější nalezení chyb. Panel Akcí zobrazuje zvýraznění chyb syntaxe a nekompatibility verze Flash Přehrávače. Také zvýrazňuje **deprecated** (neschválené) nebo již nepreferované prvky ActionScript.

Tyto volby panelu Akcí jsou dostupné jak v režimu Normálním, tak v Expert Režimu, pokud není uvedeno jinak.

Změna velikosti fontu v okně Skript:

1. Z pop-up menu panelu Akcí, vpravo nahoře, zvolte Font Size.
2. Zvolte Small, Normal nebo Large.

Import textového souboru obsahujícího ActionScript:

1. Z pop-up menu panelu Akcí, vpravo nahoře, zvolte Import from File.
2. Zvolte textový soubor obsahující ActionScript a klikněte na Open (Otevřít).

Poznámka: Skripty s chybami v syntaxi mohou být importovány pouze v Expert Režimu. V Normálním Režimu obdržíte zprávu o chybě.

Export akcí jako textového souboru:

1. Z pop-up menu panelu Akcí, vpravo nahoře, zvolte Export as File.
2. Zvolte umístění, kde bude soubor uložen a klikněte na Save.

Tisk akcí:

1. Z pop-up menu panelu Akcí, vpravo nahoře, zvolte Print. Objeví se dialog box Print.

2. Zvolte Options a klikněte na Print.

Poznámka: Vytisknutý soubor nebude zahrnovat informaci o původním Flash souboru. Je dobré zahrnout tuto informaci do **`comment`** akce ve skriptu.

Hledání textu ve skriptu, výběr volby z panelu Akcí:

- Zvolte Goto Line, pro přechod na určitý řádek ve skriptu.
- Zvolte Find pro nalezení textu.
- Zvolte Find Again pro opětovné hledání textu.
- Zvolte Replace pro nalezení a nahrazení textu.

V Režimu Expert, Replace prozkoumá celý obsah skriptu. V Normálním Režimu, Replace hledá a nahrazuje pouze text v poli parametru každé akce. Například v Normálním Režimu nemůžete nahradit všechny akce **`gotoAndPlay`** akcí **`gotoAndStop`**.

Poznámka: Pro hledání v aktuálním seznamu Akcí, použijte příkazy Find nebo Replace. Pro hledání textu v každém skriptu v celém movie použijte Movie Explorer. Více informací viz Using Flash.

Zvýraznění a kontrola syntaxe

Zvýraznění syntaxe identifikuje určité prvky ActionScriptu určitými barvami. Toto pomáhá při prevenci chyb syntaxe jako je chybné psaní klíčových slov.

Barevnostní rozlišení syntaxe:

- Keywords (Klíčová slova) a předdefinované identifikátory (například **`gotoAndStop`**, **`play`** a **`stop`**) jsou modré.
- Properties (Vlastnosti) jsou zelené.
- Comments (Komentáře) jsou fialové.
- Strings (Řetězce) obklopené dvěma uvozovkami jsou šedé.

Zapnutí a vypnutí zvýraznění syntaxe:

Zvolte Colored Syntax z pop-up menu vpravo nahoře v panelu Akcí. Kontrolní značka indikuje, že je volba zapnuta. Všechny skripty ve vašem movie budou barevně zvýrazněny.

Je dobré zkontrolovat syntaxi před exportováním movie. Chyby jsou uvedeny v okně Output. Můžete exportovat i movie, které obsahuje chybné skripty, nicméně budete varováni, že skripty obsahující chyby nebyly exportovány.

Kontrola aktuální syntaxe skriptu:

Zvolte Check Syntax z pop-up menu vpravo nahoře v panelu Akcí.

O zvýrazňování chybu

V Normálním Režimu jsou všechny chyby syntaxe znázorněny červeným pozadím v okně Skript. Toto usnadňuje nalezení problémů. Jestliže pohnete ukazatelem myši na akci s nesprávnou syntaxí, tooltip zobrazí zprávu o chybě spojenou s touto akcí. Když zvolíte akci, chybové hlášení je také zobrazeno v titulu panelu oblasti parametrů.

V Normálním Režimu jsou všechny export nekompatibility znázorněny žlutým pozadím v okně Skript. Například, jestliže export verze Flash Přehrávače je nastavena na Flash 4, ActionSctipt, který je podporován pouze Flash 5 Přehrávačem je znázorněn žlutě. Export verze je určena v dialog boxu Publish Settings.

Všechny deprecated (neschválené) akce jsou znázorněny zeleným pozadím v Toolbox listu. Neschválené akce jsou znázorněny pouze tehdy, když exportujete do Flash5 přehrávače.

Nastavení verze Flash Přehrávače pro export:

1. Zvolte File > Publish Settings.

2. Klikněte na záložku Flash.

3. Zvolte export verzi z pop-up menu Version.

Poznámka: Nemůžete vypnout znázornění syntaxe chyb.

Ukázání znázornění neschválené syntaxe:

Zvolte Show Deprecated Syntax z pop-up menu v panelu Akcí.

Kompletní seznam všech zpráv o chybách viz Dodatek C, „Hlášení o Chybách.“

Připojování akcí k objektům

Akce můžete připojit, když uživatel klikne na tlačítko, nebo přes něj přejede ukazatelem, nebo když se stahuje movie klip nebo hrací hlava dosáhne určitého snímku, akci můžete připojit také k tlačítku nebo movie klipu. Připojíte-li akci k instanci tlačítka nebo movie klipu; další instance symbolu nejsou ovlivněny. (Připojení akce ke snímku, viz „Připojování akcí ke snímkům“)

Když připojujete akci k tlačítku, musíte ji vložit dovnitř ovladače **`on (mouse event)`** a specifikovat události myši nebo klávesnice, které akci spouštějí. Když připojujete akci k tlačítku v Normálním Režimu, ovladač **`on (mouse event)`** je vložen automaticky.

Když připojujete akci k movie klipu (v expert režimu), musíte ji vložit dovnitř ovladače **`onClipEvent`** a specifikovat v klipu události, které akci spouštějí. Když připojujete akci k movie klipu v Normálním Režimu, ovladač **`onClipEvent`** je vložen automaticky.
Následující instrukce popisují jak připojit akce k objektům použitím panelu Akcí v Normálním Režimu.

Když jste připojili akci, pro testování zda funguje, použijte příkaz Control > Test Movie. Většina akcí nebude fungovat v Editovacím Režimu.

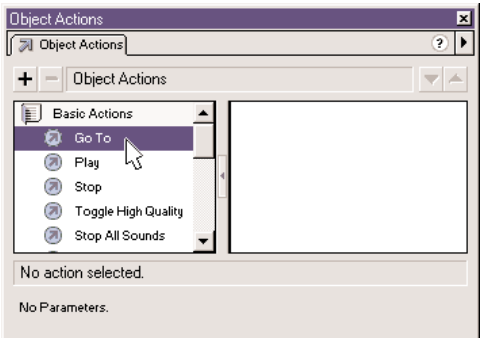
Připojení akce k tlačítku nebo movie klipu:

1. Zvolte tlačítko nebo movie klip instanci a zvolte Window > Action.

   Jestliže volbou není tlačítko, movie klip instance nebo snímek, nebo jestliže volba zahrnuje více objektů, panel Akce je matný.

2. Zvolte Normální Režim z pop-up menu vpravo nahoře v panelu Akce Objektu.

3. Pro připojení akce udělejte jedno z následujících:
   - Klikněte na složku Action v Toolbox listu na levé straně panelu Akcí. Dvakrát klikněte na akci pro její přidání do seznamu akcí na pravé straně panelu.
   - Přetáhněte akci z Toolbox listu do seznamu akcí.
   - Klikněte na tlačítko Add (+) a zvolte akci z pop-up menu.
   - Použijte klávesové zkratky zapsané vedle každé akce v pop-up menu.



4. V polích Parametry ve spodní části panelu zvolte potřebné parametry pro akci.

Parametry se liší v závislosti na akci, kterou zvolíte. Detailní informace o požadovaných parametrech pro každou akci viz Kapitola 7, „ActionScript Slovník“. Pro vložení Cílové cesty pro movie klip do pole Parametr klikněte na tlačítko Target Path v dolním pravém rohu panelu Akce. Více informací viz Kapitola 4, „Práce s Movie Klipy.“

5. Opakujte kroky 3 a 4 pro připojení potřebných dodatečných akcí.

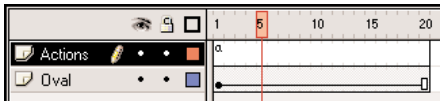
Testování akce objektu:
Zvolte Control > Test Movie.

Připojování akcí ke snímkům

Abyste přiměli movie udělat něco, když dosáhne klíčového snímku, připojíte akci snímku do klíčového snímku. Například pro vytvoření smyčky v Časové ose mezi snímky 10 a 20, byste připojili následující akci snímku do snímku 20:

```
gotoAndPlay ( 10 ) ;
```

Je dobré umístit akce símku do oddělené vrstvy. Snímky s akcemi zobrazí malé a v Časové ose.



Po připojení akce, zvolte Control > Test Movie pro testování funkčnosti. Většina akcí nefunguje v Editovacím Režimu.

Následující instrukce popisují, jak připojit akce snímku pomocí použití panelu Akcí v Normálním Režimu. (Informace o připojování akcí k tlačítku nebo movie klipu viz „Připojování akce nebo metody“.)

Připojení akce ke klíčovému snímku:

1. Zvolte klíčový snímek v Časové ose a zvolte Window > Action.

Jestliže zvolený snímek není klíčový, akce je připojena k předchozímu klíčovému snímku. Jestliže volbou není snímek, nebo jestliže volba zahrnuje více klíčových snímků, panel Akce je matný.

2. Zvolte Normální Režim z pop-up menu vpravo nahoře v panelu Akce Snímku.

3. Pro připojení akce udělejte jedno z následujících:
   - Klikněte na složku Action v Toolbox listu na levé straně panelu Akcí. Dvakrát klikněte na akci pro její přidání do seznamu akcí na pravé straně panelu.
   - Přetáhněte akci z Toolbox listu do seznamu akcí.
   - Klikněte na tlačítko Add (+) a zvolte akci z pop-up menu.
   - Použijte klávesové zkratky zapsané vedle každé akce z pop-up menu.
   - V polích Parametry v horní části panelu zvolte potřebné parametry pro akci.

4. Pro připojení dodatečných akcí zvolte další klíčový snímek a opakujte krok3.

Testování akce snímku:

Zvolte Control > Test Movie.

KAPITOLA 2

Tvorba Skriptů v ActionScriptu

Při tvorbě skriptů můžete zvolit různou úroveň složitosti a použití detailu. Pro jednoduché akce používejte panel Akcí v Normálním Režimu a tak vlastně vytvářejte skripty výběrem z možností obsažených v menu a seznamech akcí. Nicméně, jestliže chcete použít ActionScript pro tvorbu složitějších skriptů, musíte pochopit jazyk ActionSctiptu a jeho funkčnost.

Podobně jako jiné skriptovací jazyky, je ActionSctipt tvořen komponenty, jako jsou předdefinované objekty a funkce a také umožňuje vytvářet vlastní objekty a funkce. ActionScript dodržuje svá vlastní pravidla syntaxe, rezervuje si klíčová slova, poskytuje operátory a umožňuje používat proměnné pro ukládání a znovu získání informací.

Syntaxe a styl ActionSctiptu se velice podobají Java Scriptu. Flash 5 konvertuje ActionScripty napsané v jakékoliv předchozí verzi Flash.

Použití syntaxe ActionScriptu

ActionScript má pravidla gramatiky a interpunkce, určující používání znaků a slov k tvorbě skriptů a pořadí v jakém mohou být napsány. Například v angličtině tečka končí větu. V ActionSctiptu končí větu středník.

Následují všeobecná pravidla, která se uplatňují v celém ActionScriptu. Většina termínů ActionScriptu má také své vlastní individuální požadavky; pokud se týče pravidel pro specifické termíny, podívejte se na ně v Kapitole 7, „ActionScript Slovník.“

Dot syntaxe (Tečková syntaxe)

V ActionScriptu je tečka (.) používána k určení vlastností nebo metod, vztahujících se k objektu nebo movie klipu. Je také používána pro identifikaci cílové cesty k movie klipu nebo proměnné. Výraz dot syntaxe začíná jménem objektu nebo movie klipu následovaných tečkou a končí vlastností, metodou, nebo proměnnou, kterou chcete specifikovat.

Například vlastnost **_x** movie klipu indikuje pozici movie klipu po ose **x** na Scéně. Výraz **ballMC._x** referuje o vlastnosti **_x** movie klip instance **ballMC**.

Dalším příkladem je vlastnost **submit**, která je nastavena v movie klipu **form**, který je uhnízděn uvnitř movie klipu **shoppingCart**.

Výraz **shoppingCart.form.submit = true** nastavuje proměnnou **submit**, instance **form**, na **true**.

Znázornění výrazu a metody objektu nebo movie klipu je naprosto shodné. Například metoda **play** instance **ballMC** posune hrací hlavu v Časové ose **ballMC** jako v následujícím příkazu:

```
ballMC.play();
```

Dot syntaxe také používá dva speciální aliasy, **_root** a **_parent**. Alias **_root** odkazuje na hlavní Časovou osu. Alias **_root** se používá k vytvoření absolutní cílové cesty. Například následující příkaz volá funkci **buildGameBoard** v movie klipu **functions** na hlavní Časové ose:

```
_root.functions.buildGameBoard ();
```

Alias **_parent** se používá jako odkaz na movie klip, ve kterém je uhnízděn aktuální movie klip. Také se **_parent** používá k vytvoření relativní cílové cesty. Například, jestliže movie klip **dog** je uhnízděn uvnitř movie klipu **animal**, následující příkaz instance **dog** říká rodičovskému klipu **animal**, aby zastavil:

```
_parent.stop();
```

Viz Kapitola 4, „Práce s Movie Klipy.“

Slash syntaxe

Slash syntaxe byla používána ve Flash 3 a 4 pro určování cílové cesty k movie klipu nebo proměnné. Tato syntaxe je sice stále podporována Přehrávačem Flash 5, ale její používání není doporučováno. Ve slash syntaxi jsou pro určení cesty k movie klipu nebo proměnné, místo teček používána slashes (lomítka). Při určení proměnné předchází proměnnou dvojtečka:

```
myMovieClip/childMovieClip:myVariable
```

Stejná cílová cestu napsaná v dot syntaxi:

```
myMovieClip.childMovieClip.myVariable
```

Slash syntaxe byla nejčastěji používána s akcí **tellTarget**, jejíž použití již také není doporučováno.

Poznámka: Akce **with** je nyní preferována před akcí **tellTarget**, protože je kompatibilnější s dot syntaxí. Více informací viz Kapitola 7, „ActionScript Slovník.“

Složené závorky

Příkazy ActionScriptu jsou seskupeny do bloků složenými závorkami ({}), jako v následujícím skriptu:

```
on(release){
    myDate = new Date();
    currentMonth = myDate.getMonth();
}
```

Viz „Použití akcí“.

Středníky

Příkaz ActionScriptu je ukončen středníkem, ale pokud středník zapomeneme zapsat, Flash bude stále úspěšně se skriptem pracovat. Například následující výrazy jsou zakončeny středníky:

```
column = passedDate.getDay();
row = 0;
```

Stejné příkazy by mohly být napsány bez zakončení středníky:

```
column = passedDate.getDay()
row = 0
```

Závorky

Když definujete funkci, umístěte jakékoliv argumenty do závorek:

```
function myFunction (name,age,reader){
    ...
}
```

Když voláte funkci, vložte všechny argumenty k ní přiřazené do zavorek:

```
myFunction ("Steve",10,true);
```

Závorky také můžete použít pro předepsání priorit přednosti, nebo pro usnadnění čtení ActionScript výrazů. Viz „Priority operátorů“.

V dot syntaxi se také používají pro ohodnocení výrazu na levé straně od tečky. Například v následujícím příkazu závorky způsobí, že **new Color(this)** bude ohodnocen a vznikne objekt s novou barvou:

```
onClipEvent (enterFrame){
    (new Color(this)).setRGB(0xffffff);
}
```

Kdybyste nepoužili závorky a kód byl ohodnocen shodně, museli byste do něj přidat nějaké příkazy:

```
onClipEvent (enterFrame){
    myColor = new Color(this);
    myColor.setRGB(0xffffff);
}
```

Malá a velká písmena

V ActionScriptu jsou case citlivá (citlivá na malá a velká písmena) pouze klíčová slova; ve zbytku ActionScriptu můžete používat malá i velká písmena, podle vlastního uvážení. Například následující výrazy jsou ekvivalentní:

```
cat.hilite = true;
CAT.hilite = true;
```

Nicméně, je dobrým zvykem pro usnadnění identifikace jmen funkcí a proměnných při čtení a orientaci v kódu ActionScriptu, dodržovat odpovídající konvence psaní velkých a malých písmen, používané v této knize,.

Jestliže nepoužíváte správně psaní velkých a malých písmen u klíčových slov, váš scénář bude obsahovat chyby. Když je zapnuta Barevná Syntaxe v panelu Akcí, klíčová slova napsaná správnými písmeny jsou modrá. Více informací viz „Klíčová slova“ a „Zvýraznění a kontrola syntaxe“.

Komentáře

Pro popis a sledování průběhu akcí a pro přidávání poznámek ke snímkové nebo tlačítkové akci v panelu Akce, použijte výraz comment.

Jestliže pracujete v kolektivu, nebo někomu poskytujete ukázky skriptu, jsou komentáře také užitečné pro poskytování informací dalším vývojářům.

Když zvolíte akci **comment**, jsou do skriptu vloženy znaky //. Dokonce i jednoduché skripty jsou s poznámkami o jejich tvorbě snadněji pochopitelné:

```
on(release){
    // vytvořit nový objekt Datum
    myDate = new Date();
    currentMonth = myDate.getMonth();
    // konvertovat číslo měsíce na jméno měsíce
    monthName = calcMonth(currentMonth);
    year = myDate.getFullYear();
    currentDate = myDate.getDat ();
}
```

Komentáře se v okně Skript objeví růžově znázorněny. Mohou mít jakoukoliv délku bez ovlivnění velikosti exportovaného souboru a nemusí dodržovat pravidla syntyxe ActionScriptu, nebo klíčová slova.

Klíčová slova

ActionScript rezervuje slova pro specifické použití uvnitř jazyka, takže je nemůžete použít jako proměnné, funkce nebo popisy jmen. V následující tabulce je seznam všech ActionScript klíčových slov:

Více informací o speciálních klíčových slovech viz Kapitola 7, „ActionScript Slovník.“

break	for	new	var
continue	function	return	void
delete	if	this	while
else	in	typeof	with

Konstanty

Konstanta je vlastnost, jejíž hodnota se nikdy nemění. Konstanty jsou zapsány v Actions toolbox a v Kapitole 7, „ActionScript Slovník,“ všechny velkými písmeny.

Například konstanty **BACKSPACE**, **ENTER**, **QUOTE**, **RETURN**, **SPACE** a **TAB** jsou vlastnosti objektu Key a odkazují na klávesy klávesnice. Pro testování, zda uživatel tiskne klávesu Enter, použijte následující příkaz:

```
if(keycode() == Key.ENTER){
    alert = "Are you ready to play?"
    controlMC.gotoAndStop(5);
}
```

O typech dat

Typ dat popisuje druh informace, kterou může obsahovat proměnná nebo prvek. Existují dva typy dat: primitivní a odkazující. Primitivní typy dat – string, number a Boolean - mají konstantní hodnotu a proto mohou obsahovat aktuální hodnotu prvku, který reprezentují. Odkazující typy dat - movie klip a objekt - mají hodnotu, která se může měnit a proto obsahují odkazy na aktuální hodnotu prvku. Proměnné obsahující primitivní typy dat se chovají odlišně v určitých situacích než ty, které obsahují odkazující typy. Viz „Použití proměnných ve skriptu“.

Každý typ dat má svá vlastní pravidla a jejich seznam je zde. Odkazy jsou zahrnuty pro typy dat, které jsou rozebrány detailněji.

String (Řetězec)

Řetězec je sekvence znaků jako jsou písmena, čísla a interpunkční znaménka. Řetězce vložíte do ActionScript výrazu jejich vnořením do jednoduché nebo dvojité citace (uvozovek). S řetězci se zachází jako se znaky, ne jako s proměnnými. Například následující výraz „L7“ je řetězec:

```
favoriteBand = "L7";
```

Pro sloučení nebo spojení dvou řetězců můžete použít operátor Add (+). ActionScript zachází s mezerami na začátku nebo konci řetězce jako s částí řetězce. Následující výraz zahrnuje mezeru za čárkou:

```
greeting = "Welcome," + firstName;
```

Ačkoliv ActionScript nerozlišuje mezi velkými a malými písmeny v odkazech na proměnné, jména instancí a popisy snímků, písmenné znaky jsou case citlivé. Například, následující dva výrazy umísťují různý text do určených textových polí proměnných, protože „Hello“ a „HELLO“ jsou písmenné řetězce.

```
invoice.display = "Hello";
invoice.display = "HELLO";
```

Pokud vkládáte do řetězce uvozovky, uveďte je zpětným lomítkem (\ ). Toto je nazváno „escaping“ znak. Jsou zde i další znaky, které mohou být prezentovány v ActionScriptu pouze ve speciálních escape sekvencích. Následující tabulka poskytuje únikové znaky ActionSctipt:

Escape sekvence	Představovaný znak
\b	Backspace (ASCII 8)
\f	Form-feed (ASCII 12)(posuv stránky)
\n	Line-feed (ASCII 10)(nový řádek)
\r	Carriage return (ASCII 13)(návrat kurzoru)
\t	Tab (ASCII 9) (tabulátor)
\"	Dvojitý znak citace
\'	Jednoduchý znak citace
\\	Backslash (zpětné lomítko)
\000 - \377	Byte specifikovaný v octal
\x00 - \xFF	Byte specifikovaná v hexadecimal
\u0000 - \uFFFF	16-ti bit Unicode znak specifikovaný v hexadecimal

Number (Čísla)

Čísla jsou data buď celočíselná, nebo s pohyblivou čárkou. S čísly můžete manipulovat pomocí aritmetických operátorů sčítání (+), odčítání (-), násobení (*), dělení (/), procento (%), inkrement (++) a dekrement (--). Pro práci s čísly můžete také použít metody předdefinovaného objektu Math. Následující příklad používá metodu **sqrt** (čtvercový kořen) pro návrat čtvercového kořene na číslo 100:

```
Math.sqrt(100);
```

Viz „Numerické operátory“.

Boolean (Booleovské)

Booleovská hodnota je taková hodnota, která je buď **true** nebo **false**. ActionScript také, pokud je to vhodné konvertuje hodnoty **true** a **false**, na 1 a 0. Booleovské hodnoty jsou nejčastěji používány s logickými operátory v těch příkazech ActionScriptu, které dělají srovnání pro kontrolu toku skriptu. Například v následujícím scénáři movie hraje, jestliže proměnná **password** je **true**:

```
onClipEvent(enterFrame) {
        if ((userName == true) && (password == true)) {
              play();
        }
}
```

Viz „Použití příkazů „if“ a „Logické operátory“.

Object (Objekt)

Objekt je soubor vlastností, znichž každá má jméno a hodnotu. Hodnota vlastnosti může být jakýkoliv typ dat. To umožňuje upravovat objekty uvnitř sebe a nebo je „uhnízdit“ jeden do druhého. Pro specifikaci objektů a jejich vlastností používejte operátor dot (.). Například v následujícím kódu je **hoursWorked** vlastností **weeklyStats**, která je vlastností **employee**:

```
employee.weeklyStats.hoursWorked
```

Pro dosažení a manipulaci se specifickými druhy informací, můžete použít ActionScriptem předdefinované objekty. Například objekt Math má metody vykonávající matematické operace s čísly, která jste v nich definovali. Teto příklad používá metodu **sqrt**:

```
squareRoot = Math.sqrt(100);
```

Objekt ActionScriptu MovieClip, má metody umožňující kontrolu instancí movie klip symbolu na Scéně. Tento příklad používá metody **play** a **nextFrame**:

```
mcInstanceName.play();
mc2InstanceName.nextFrame();
```

Pro lepší organizaci informací v animaci si můžete vytvořit vlastní objekty. Při přidávání interaktivity do animace, je třeba spousta různých druhů informací: například jméno uživatele, rychlost míče, jména položek v nákupní kartě, počet natažených snímků, uživatelův zip kód a např. skutečnost, která klávesa byla naposledy stisknuta. Vytvoření speciálních objektů umožňuje organizovat tyto informace do skupin, usnadnit tvorbu a znovupoužití skriptů. Více informací viz „Použití speciálních objektů“.

Movie clip (Movie klip)

Movie klipy jsou symboly, které mohou hrát svou animaci v další Flash animaci. Jsou to pouze typy dat odkazující na grafický prvek. Typy dat movie klipu umožňují kontrolu symbolů movie klipu používajících metody MovieClip objektu. Metody můžete volat použitím operátoru dot (.):

```
myClip.startDrag(true);
parentClip.childClip.getURL(„http://www.macromedia.com/support/"+product);
```

O proměnných

Variable (Proměnná) je schránka, která obsahuje informaci. Samotná schránka je stále stejná, avšak její obsah se může měnit. Změnou hodnoty proměnné v průběhu animace můžete nahrát a uložit informace o tom, co uživatel udělal, nahrát hodnoty, které se změnily během přehrávání animace, nebo ohodnotit zda je nějaká podmínka pravdivá nebo ne.

Je dobré vždy připojit proměnné hodnotu hned při její první definici. Toto nazýváme zavedení (deklarace) proměnné a je často prováděno v prvním snímku animace. Zavedení proměnné, usnadňuje v průběhu animace sledování a porovnávání hodnoty proměnné.

Proměnné mohou obsahovat jakékoliv typy dat: číslo, řetězec, Booleánské, objekt nebo movie klip. Typy dat, které proměnná obsahuje ovlivňují, jak se mění hodnota proměnné, při jejím zavedení do skriptu.

Typy dat a informací, které můžete ukládat do proměnné jsou například: URL, jméno uživatele, výsledek matematické operace, počet, kolikrát se objevila nějaká událost nebo zda bylo kliknuto na tlačítko. Každé movie a movie klip instance má své vlastní nastavení proměnných, s tím, že každá proměnná má svoji vlastní hodnotu nezávislou na proměnných v jiných movies nebo movie klipech.

Pojmenovávání proměnných

Jméno proměnné musí dodržovat tato pravidla:

- Musí to být identifikátor.
- Nemůže to být klíčové slovo nebo Booleovské funkce (**`true`** nebo **`false`**).
- Musí být unikátní ve svém rozsahu. (Viz „Rozsah proměnné“)

Zápis proměnné

Ve Flashi nemusíte výslovně (explicitně) definovat proměnnou tak, že obsahuje buď číslo, řetězec nebo jiný typ dat. Flash určuje typ data proměnné až když je proměnná připojena:

```
x = 3;
```

Ve výrazu **`x = 3`**, Flash ohodnotí prvek na pravé straně operátoru a určí, zda je to typ čísla. Později se typ x může změnit; například **`x = „ahoj"`** změní typ **`x`** na řetězec. Proměnná, které nebyla přiřazena žádná hodnota má typ undefined (nedefinovaná).

ActionScript convertuje typy dat automaticky, pokud to výraz vyžaduje. Například, když vložíte hodnotu do akce **`trace`**, **`trace`** automaticky konvertuje hodnotu na řetězec a posílá ji do okna Output. Ve výrazech s operátory konvertuje ActionScript typy dat podle potřeby; například jestliže je použit řetězec, operátor + očekává, že dalším operandem bude řetězec:

```
„Další v řadě, je číslo"+7
```

ActionScript konvertuje číslo 7 na řetězec „7“ a přidá ho nakonec prvního řetězce:

```
„Další v řadě, je číslo 7"
```

Při odstraňování závad ve skriptech, je užitečné určit typy dat ve výrazu nebo proměnné k pochopení, proč se skript chová určitým způsobem. Můžete to udělat pomocí operátoru **`typeof`**:

```
trace(typeof(variableName));
```

Pro konvertování řetězce na numerickou hodnotu použijte funkci **`Number`**. Pro konvertování numerické hodnoty na řetězec použijte funkci **`String`**. Viz Kapitola 7, „ActionScript Slovník“.

Rozsah proměnné

„Rozsah“ proměnné odkazuje na oblast, ve které proměnná existuje a může na ni být odkazováno. Proměnné v ActionScriptu mohou být buď globální nebo lokální. Globální proměnná je sdílena všemi Časovými osami; lokální proměnná je dostupná pouze uvnitř svého vlastního bloku kódu (mezi složenými závorkami).

Pro deklarování lokální proměnné uvnitř skriptu, můžete použít příkaz **var**. Například proměnné i a j jsou často používány jako počítadla smyček. V následujícím příkladě je i použito jako lokální proměnná; existuje pouze uvnitř funkce **makeDays**:

```
function makeDays(){
   var i
   for(i = 0; i < monthArray [month]; i++){

      _root.Days.attachMovie("DayDisplay", i, i + 2000);

      _root.Days [i].num = i + 1;
      _root.Days [i]._x = column *_root.Days [i]._width;
      _root.Days [i]._y = row *_root.Days [i]._height;

      column = column + 1;

      if (column == 7){

         column = 0;
         row = row + 1;
      }
   }
}
```

Lokální proměnné také mohou pomoci při prevenci proti kolizím jmen (shodě jmen), které mohou způsobit chyby ve v animaci. Například, jestliže používáte **name** jako lokální proměnnou, můžete ji použít pro uložení jména uživatele v jednom kontextu a jméno movie klip instance v jiném kontextu; protože tyto proměnné poběží v oddělených prostorech, nedojde zde k žádná kolizi.

V praxi je dobré používat lokální proměnné v těle funkce tak, aby funkce mohla fungovat jako nezávislá část kódu. Lokální proměnnou lze změnit pouze uvnitř jejího bloku kódu. Jestliže výraz ve funkci používá globální proměnnou, příkaz zvenčí funkce může změnit jeho hodnotu a ta změní celou funkci.

Deklarace (zavádění) proměnné

Pro deklaraci globálních proměnných použijte akci **setVariable** nebo operátor (=). Oběma metodami dosáhnete stejných výsledků.

Pro deklaraci lokálních proměnných použijte příkaz **var** uvnitř těla funkce. Lokální proměnné jsou seskupeny do bloku a jejich účinnost končí na konci bloku. Lokálním proměnným nedeklarovaným uvnitř bloku končí účinnost na konci jejich skriptu.

Poznámka: Akce **call** také vytváří novou lokální proměnnou, deklarovanou do volaného skriptu. Jestliže volaný skript skončí, deklarovaná lokální proměnná zmizí. Nicméně, tato akce není doporučována, protože akce **call** byla nahrazena akcí **with**, která je kompatibilnější s dot syntaxí.

Pro testování hodnoty proměnné použijte akci **trace**, která posílá hodnoty do okna Output. Například, **trace(hoursWorked)** pošle v testovacím režimu hodnotu proměnné **hoursWorked** do okna Output. Kontrolovat a nastavit hodnoty proměnné v testovacím režimu můžete také v Debuggeru (Odstraňovači závad). Více informací viz Kapitola 6, „Odstraňování závad ActionScriptu.“

Použití proměnných ve skriptu

Proměnnou musíte ve skriptu deklarovat ještě před jejím použitím ve výrazu. Jestliže použijete nedeklarovanou proměnnou, jako v následujícím příkladě, hodnota proměnné bude **undefined** (nedefinována) a skript vygeneruje chybu:

```
getURL(myWebSite);
myWebSite = "http://www.shrimpmeat.net";
```

Výraz deklarující proměnnou **myWebSite** musí být první, aby proměnná v akci **getURL** mohla být nahrazena její hodnotou.

Hodnotu proměnné můžete ve skriptech změnit několikrát. Datové typy které proměnná obsahuje, ovlivňují, jak a kdy se proměnná změní. Základní typy dat (řetězce a čísla) jsou nahrazeny hodnotou. To znamená, že aktuální obsah proměnné je přiřazen k proměnné.

V následujícím příkladě je **x** nastaveno na 15 a hodnota je zkopírována do **y**. Když je **x** změněno na 30, hodnota **y** zůstává 15, protože **y** nehledá hodnotu **x** , ale obsahuje již dříve přiřazenou hodnotu **x**.

```
var x =  15;
var y =  x;
var x = 30;
```

V dalším příkladě, proměnná **in** má původní hodnotu 9, aktuální hodnota je vložena do funkce **sqrt** a nově získaná hodnota je 3:

```
function sqrt(x){
   return x * x;
}

var in = 9;
var out = sqrt (in);
```

Hodnota proměnné **in** se nezmění.

Datový typ objekt může obsahovat tak velké množství mnohdy složitých informací, že proměnná tohoto typu v sobě neuchovává samotnou proměnnou, ale pouze odkaz na ni. Tato reference je něco jako alias, který se odkazuje na obsah proměnné. Když proměnná potřebuje znát svou hodnotu, vyžádá si odkaz příslušný obsah a vrátí odpověď, aniž by se hodnota přesunula do proměnné.

Následuje příklad odkazu na obsah:

```
var myArray = ["tom","dick"];
var newArray = myArray;
myArray [1] == "jack";
trace(newArray);
```

Předchozí kód vytváří objekt Array nazvaný **myArray**, se dvěma prvky. Deklarovaná proměnná **newArray** má hodnotu **myArray**. Když je druhý prvek **myArray** změněn, ovlivní to každou proměnnou s odkazem na něj. Akce **trace** by posílala **[„tom", „jack"]** do okna Output.

V dalším příkladě obsahuje **myArray** objekt Array, takže je poslán funkci **zeroArray**. Funkce **zeroArray** změní obsah pole v **myArray**.

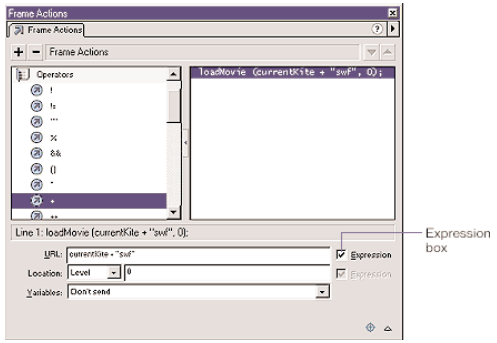
```
function zeroArray (array){
   var i;
   for (i = 0; i < array.length; i++){
      array [i] == 0;
   }
}

var myArray = new Array();
myArray [0] == 1;
myArray [1] == 2;
myArray [2] == 3;

var out = zeroArray(myArray)
```


Funkce zeroArray přijme objekt Array jako argument a nastaví všechny prvky tohoto pole na 0. Může pole změnit, protože pole obsahuje referenci.

Odkazy odvolávající se na jiné objekty než jsou klipy animace, se nazývají **hard references** (tvrdé odkazy), protože pokud na objekt existuje odkaz, není možno jej smazat. Odkaz na klip animace je zvláštní druh odkazu a nazývá se **soft references** (měkký odkaz). Měkké odkazy nevyžadují, aby daný objekt existoval. Pokud je klip animace zrušen akcí jako je například **removeMovieClip**, nebudou veškeré odkazy na něj odkázané fungovat.

Použití operátorů pro manipulaci s hodnotami ve výrazech

Expression (Výraz) je jakýkoliv příkaz, který Flash ohodnotí a přidělí mu hodnotu. Výraz můžete vytvořit kombinací operátorů a hodnot, nebo voláním funkce. Při psaní výrazu v panelu Akce v Normálním Režimu se ujistěte, že box Expression je v panelu Parametrů zaškrtnutý, jinak bude pole místo výrazu obsahovat písmennou hodnotu řetězce.



Operátory jsou znaky, které určují, jak kombinovat, srovnávat nebo upravovat hodnoty výrazu. Prvky, mezi kterými operátor leží a se kterými manipuluje se nazývají **operandy**. Například v následujícím výrazu operátor + přidává numerickou hodnotu k hodnotě proměnné **foo**; **foo** a **3** jsou operandy:

```
foo + 3
```

Tato část popisuje všeobecná pravidla u běžných typů operátorů. Detailnější informace o každém operátoru stejně tak jako o speciálních operátorech nespadajících do těchto kategorií, viz Kapitola 7, „ActionScript Slovník."

Priorita operátorů

Při použití dvou nebo více operátorů v jednom výrazu, mají některé operátory přednost před jinými. ActionScript dodržuje přesnou hierarchii k určování priority operátorů. Například, násobení je vždy provedeno před sčítáním, ale položky v závorkách mají přednost před násobením. Bez závorek ActionScript provádí násobení jako první:

```
celkem = 2 + 4 * 3 ;
```

Výsledek je 14.

Když závorky obsahují operaci sčítání, ActionScript vykoná nejprve sčítání:

```
celkem = ( 2 + 4 ) * 3 ;
```

Výsledek je 18.

Tabulka všech operátorů a jejich přednosti viz Dodatek A, „Priorita Operátorů a Pořadí vyhodnocování.“

Pořadí vyhodnocování operátorů

Jestliže mají dva nebo více operátorů stejnou prioritu, pořadí ve kterém budou vykonány určuje jejich pořadí vyhodnocování, které může být buď L - zleva doprava nebo R - zprava doleva. Například operátor násobení má pořadí vyhodnocování L - zleva doprava, proto následující dva výrazy jsou ekvivalentní:

```
celkem = 2 * 3 * 4;
celkem = (2 * 3) * 4;
```

Tabulka všech operátorů a jejich pořadí vyhodnocování viz Dodatek A, „Priorita Operátorů a Pořadí vyhodnocování.“

Aritmetické (numerické) operátory

Aritmetické (numerické) operátory sčítají, odčítají, násobí, dělí a provádí další aritmetické operace. Závorky a znak minus jsou také aritmetické operátory. Následuje tabulka numerických operátorů ActionScriptu:

Operátor	Vykonaná operace
+	Sčítání
*	Násobení
/	Dělení
%	Procento
-	Odčítání
++	Inkrement (zvýšení)
--	Dekrement (snížení)

Srovnávací (relační) operátory

Srovnávací operátory porovnávají hodnoty výrazů a dávají Boolánskou hodnotu (**true** nebo **false**). Tyto operátory jsou nejběžněji používány ve smyčkách a v podmínkových výrazech. V následujícím příkladě, jestliže proměnná **skore** je **100**, natáhne se určitý .swf soubor, jinak se natáhne jiný .swf soubor:

```
if (skore == 100){
   loadMovie("vitez.swf",5);
}else {
   loadMovie("porazeny.swf",5);
   }
```

Následuje tabulka srovnávacích (relačních) operátorů:

Operátor	Vykonaná operace
<	Menší než
>	Větší než
<=	Menší nebo rovno
>=	Větší nebo rovno

Řetězcové operátory

Operátor **+** , v řetězcích spojuje dva řetězcové operandy. Například, následující příkaz přidává:

```
"Congratulations," to "Donna!":
"Congratulations," + "Donna!"
```

Výsledek je **"Congratulations, Donna!"**. Jestliže je pouze jeden operand operátoru + řetězec, Flash konvertuje další operand také na řetězec.

Srovnávací operátory **>**, **>=**, **< a <=** mají při operacích v řetězcích speciální určení. Tyto operátory porovnávají dva řetězce a určují, který je první v abecedním pořadí. Srovnávací operátory porovnávají řetězce pouze tehdy, jestliže oba operandy jsou řetězce. Jestliže je pouze jeden z operandů řetězec, ActionScript konvertuje oba operandy na čísla a vykoná numerické srovnání.

Poznámka: Psaní dat ActionScriptu ve Flash 5 umožňuje použít stejné operátory pro různé druhy dat. Při exportu animace jako Flash 4 soubor, není nutné používat Flash 4 řetězcové operátory (například, **eq**, **ge**, a **lt**).

Logické operátory

Logické operátory porovnávají Booleánské (**true** a **false**) hodnoty a dávají třetí Booleánskou hodnotu. Například, jestliže dva operandy mají hodnotu **true**, logický operátor AND (**&&**) dá **true**. Jestliže jeden nebo oba operandy mají hodnotu **true**, logický operátor OR (**||**) dá **false**. Logické operátory jsou často používány ve spojení se srovnávacími operátory pro určení podmínky akce **if**. Například v následujícím scénáři, jestliže jsou oba výrazy **true**, akce **if** bude provedena:

```
if ((i > 10) && (_framesloaded > 50)){
   play();
}
```

Následuje tabulka logických operátorů:

Operátor	Vykonaná operace
&&	Logické AND
||	Logické OR
!	Logické NOT

Bitové operátory

Přestože v Actionscriptu jsou všechna čísla s pohyblivou čárkou, bitové operátory vyžadují číselné operandy s celočíselnou hodnotou. Operují s těmito celočíselnými operandy za použití 32-bitových celých čísel, se kterými se snadněji pracuje. Přesnost vykonané bitové operace závisí na operátoru, avšak všechny bitové operace ohodnocují každou číslici s pohyblivou čárkou odděleně pro počítání nové hodnoty.

Následuje tabulka bitových operátorů ActionScriptu:

Operátor	Vykonaná operace
&	Bitové And
|	Bitové Or
^	Bitové Xor
~	Bitové Not
<<	Posuv vlevo
>>	Posuv vpravo se znaménkem
>>>	Posuv vpravo s doplněním nulou

Tyto operátory jsou používány pro nízkoúrovňovou manipulaci s dvojkovými čísly a nejsou při programování běžně používány. Pro lepší pochopení je vhodné se obeznámit s dvojkovými (binárními) čísly a binární reprezentací desítkových celých čísel.

Srovnávací a přiřazovací operátory

K určování, zda jsou hodnoty nebo identity dvou operandů sobě rovny, používejte operátor rovnost (**==**). Toto srovnání dává Booleovskou (**true** nebo **false**) hodnotu. Jestliže jsou operandy řetězce, čísla nebo Booleovské hodnoty, jsou porovnávány hodnotou. Jestliže jsou operandy objekty nebo pole, jsou porovnávány odkazem.

Stanovení hodnoty proměnné pomocí operátoru přiřazení (**=**):

```
password = "Sk8tEr";
```

Tento operátor lze také použít pro stanovení několika proměnných ve stejném výrazu. V následujícím výrazu hodnota b je přiřazena hodnotám c a d:

```
a = b = c = d;
```

Pro kombinování operací používejte složené přiřazovací operátory. Složené operátory se vykonají na obou operandech a potom určí novou hodnotu prvního operandu. Například následující výrazy jsou ekvivalentní:

```
x += 15;
x = x + 15;
```

Následuje tabulka srovnávacích a přiřazovacích operátorů:

Operátor	Vykonaná operace
==	Rovnost
!=	Nerovnost
=	Přiřazení
+=	Přidání a přiřazení
-=	Odečtení a přiřazení
*=	Násobení a přiřazení
%=	Procento a přiřazení
/=	Dělení a přiřazení
<<=	Bitový posun vlevo a přiřazení
>>=	Bitový posuv vpravo a přiřazení
>>>=	Bitový posuv vpravo s doplněním nulou a přiřazení
^=	Bitové Xor a přiřazení
|=	Bitové Or a přiřazení
&=	Bitové And a přiřazení

Operátory tečka a přístupové pole

Operátor tečka (**.**) a operátor přístupové pole (**]**) lze používat pro dosažení jakýchkoliv předdefinovaných nebo speciálních vlastností objektu ActionScriptu, včetně vlastností movie klipu.

Operátor tečka používá jméno objektu na levé straně a jméno vlastnosti nebo proměnné na pravé straně. Jméno vlastnosti nebo proměnné nemůže být řetězec nebo proměnná, která je ohodnocována jako řetězec; musí to být identifikátor. Následující příklady používají tečkový operátor:

```
year.month = "June";
year.month.day = 9;
```

Operátor tečka a operátor přístupové pole vykonávají stejnou roli, ale operátor tečka bere identifikátor jako jeho vlastnost a operátor přístupové pole ohodnocuje jeho obsahy na jméno a potom určuje hodnoty této pojmenované vlastnosti. Například následující dva řádky kódů přisuzují stejnou proměnnou **velocity** v movie klipu **rocket**:

```
rocket.velocity;
rocket [„velocity"];
```

Operátor přístupové pole můžete použít pro dynamické nastavení a získání jmen instancí a proměnných. Například v následujícím kódu je výraz uvnitř operátoru **[]** ohodnocen a výsledek ohodnocení je použit jako jméno proměnné, která bude získána z movie klipu **name**:

```
name [ „mc" + i ]
```

Jestliže ovládáte slash syntaxi (Flash 4), můžete udělat stejnou věc použitím funkce **eval**:

```
 eval(„mc"& i);
```

Operátor přístupové pole může být také použit na levé straně určujícího příkazu. Toto vám umožňuje dynamicky nastavit jména instancí, proměnných a objektů:

```
name[index] = „Gary";
```

Opět je to ekvivalentní k následující slash syntaxi (Flash 4):

```
Set Variable : "name:" & index = „Gary"
```

Operátor přístupové pole může také vkládat sám sebe při simulaci vícerozměrných polí.

```
chessboard [row] [column]
```

Toto je ekvivalentní k následující slash syntaxi:

```
eval („chessboard/" & row & ":" & column)
```

Poznámka: Jestliže chcete psát skripty které jsou kompatibilní s Flash 4 Přehrávačem, můžete použít akci **eval** s operátorem **add**.

Používání akcí

Akce jsou příkazy nebo povely ActionScriptu. Několik akcí určených pro stejný snímek nebo objekt vytváří skript. Akce mohou hrát nezávisle na sobě, jako v následujících příkazech:

```
swapDepths(„mc1","mc2");
gotoAndPlay (15);
```

Také můžete „uhnízdit" akce použitím jedné akce uvnitř jiné; toto umožňuje akcím navzájem se ovlivňovat. V následujícím příkladě akce **if**, říká akci **gotoAndPlay** kdy se má vykonat:

```
if (i > 25){
   gotoAndPlay (10);
}
```

Akce mohou posunovat hrací hlavu po Časové ose (**gotoAndPlay**), kontrolovat tok skriptu vytvářením smyček (**do**…**while**), nebo logických podmínek (**if**), nebo vytvořit nové funkce a proměnné(**function**, **setVariable**). Následuje tabulka všech ActionScript akcí:

Action	Action
break	onClipEvent
call	play
comment	playFrame
continue	prevScene
delete	print
do…while	printAsBitmap
duplicateMovieClip	removeMovieClip
else	return
else if	setVariable
evalute	setProperty
for	startDrag
for…in	stop
fsCommand	stopAllSounds
function	stopDrag
getURL	swapDepths
gotoAndPlay, gotoAndStop	tellTarget
if	toggleHighQuality
ifFrameLoaded	stopDrag
include	trace
loadMovie	unloadMovie
loadVariables	var
nextFrame, nextScene	while
on	

Syntaxe a použití příkladů každé akce viz Kapitola 7, „ActionScript Slovník.“

Poznámka: V této knize je termín ActionScriptu **action** (akce) synonymum s JavaScript termínem **statement** (příkaz).

Zápis cílové cesty

Při použití akce, která kontroluje movie klip nebo nataženou animaci, musíte specifikovat jeho jméno a jeho adresu nazvanou **target path** (cílová cesta). Následující akce mají jednu nebo více cílových cest jako argumenty:

- `loadMovie`
- `loadVariables`
- `unloadMovie`
- `setProperty`
- `startDrag`
- `duplicateMovieClip`
- `removeMovieClip`
- `print`
- `printAsBitmap`
- `tellTarget`

Například akce `loadMovie` má argumenty **URL**, **Location** a **Variables**. URL je umístění animace, kterou chcete natáhnout. Umístění je cílová cesta, do které bude animace natažena.

```
loadMovie(URL ,Location ,Variables);
```

Poznámka: Argument Proměnné není v tomto příkladě požadován.

Následující příkaz natahuje URL http://www.mySite.com/myMovie.swf do instance `bar` na hlavní Časové ose `( _root);` `_root.bar` je cílová cesta;

```
loadMovie("http://www.mySite.com/myMovie.swf",_root.bar);
```

V ActionScriptu identifikujete movie klip jménem jeho instance. Například v následujícím výrazu vlastnost `_alpha` movie klipu pojmenovaného `star` je nastavena na 50% viditelnost:

```
star._alpha = 50;
```


Pojmenování movie klipu jménem instance:

1. Vyberte movie klip na Scéně.
2. Zvolte Window > Panels > Instance.
3. Vložte jméno instance v poli Name.

Identifikace natažené animace:

Použijte `_levelX`, kde `X` je číslo úrovně určené v akci `loadMovie`, která animaci natahuje.

Například animace natažená do úrovně 5 má jméno instance `_level5`. V následujícím příkladu je animace natažena do úrovně 5 a jeho viditelnost je nastavena na `false`:

```
onClipEvent(load){
    loadMovie("myMovie.swf",5);
}
onClipEvent(enterFrame){
    _level5._visible = false;
}
```

Vložení cílové cesty klipu:

Klikněte na tlačítko Insert Target Path v panelu Akcí a ze seznamu, který se objeví zvolte movie klip.

Více informací o psaní cílových cest viz Kapitola 4, „Práce s Movie Klipy.“

Kontrola toku ve skriptech

ActionScript používá akce **if, for, while, do**...**while** a **for**...**in** pro vykonání akcí v závislosti na skutečnosti, zda existuje podmínka.

Použití příkazu "if"

Příkazy, které kontrolují, zda je podmínka **true** nebo **false** začínají termínem **if**. Jestliže podmínka existuje, ActionScript vykoná příkaz který následuje. Jestliže podmínka neexistuje, ActionScript skočí na další příkaz mimo bloku kódu.

Pro optimalizaci kódu zkontrolujte nejprve podmínky.

Následující příklad testuje několik podmínek. Termín **else** if specifikuje alternativní varianty provedení, jestliže jsou předchozí podmínky false.

```
if ((password == null)||(email == null)){
   gotoAndStop("reject");
} else {
   gotoAndPlay("startMovie ");
}
```

Opakování akce

ActionScript může opakovat akci několikrát, nebo zatímco existuje určitá podmínka. Pro vytvoření smyček použijte akce **while, do**...**while, for a for...in**.

Opakování akce, zatímco existuje podmínka:
Použijte příkaz **while**.

Smyčka **while** ohodnocuje výraz a vykonává kód v těle smyčky, jestliže je výraz **true**. Pokaždé, když je v tělě příkaz vykonán, výraz je ohodnocen znovu. V následujícím příkladě se smyčka vykonává čtyřikrát:

```
i = 4
while (i > 0){
   myMC.duplicateMovieClip("newMC" + i, i);
   i--;
}
```


Příkaz **do**...**while** můžete použít k vytvoření stejného druhu smyčky jako je smyčka **while**. Ve smyčce **do**...**while** je výraz ohodnocen na spodku bloku kódu, takže smyčka vždy běží minimálně jednou:

```
i = 4
do {
   myMC.duplicateMovieClip("newMC" + i, i);
   i--;
} while (i > 0);
```

Opakování akce pomocí použití zabudovaného počítadla:
Použijte příkaz **for**.

Většina smyček používá pro kontrolu kolikrát už smyčka běžela, nějaký typ počítadla. Můžete deklarovat proměnnou a napsat příkaz, který zvyšuje nebo snižuje proměnnou pokaždé, když se smyčka vykoná. V akci **for** jsou počítadlo a příkaz, který zvyšuje počítadlo, součástí akce:

```
for (i = 4; i > 0; i--) {
   myMC.duplicateMovieClip("newMC" + i, i +10);
}
```

Pro vytvoření smyčky prostřednictvím dítěte movie klipu nebo objektu:
Použijte příkaz **for..in**.

Dítě zahrnuje další movie klipy, funkce, objekty a proměnné. Následující příklad používá akci **trace** pro tisk výsledků do okna Output:

```
myObject ={name:'Joe',age:25,city:'San Francisco'};
for (propertyName in myObject){
   trace("myObject has the property:"+propertyName +",with
the value:"+myObject [propertyName]);
}
```

Tento příklad posílá výsledky do okna Výstup:

```
myObject has the property: name,with the value: Joe
myObject has the property: age,with the value: 25
myObject has the property: city,with the value: San Francisco
```

Možná budete chtít, aby váš skript prováděl iteraci na určitém typu dítěte - např. jen na dětech klipů animace. To můžete provést pomocí **for...in** ve spojení s operátorem **typeof**.

```
for (name in myMovieClip) {
    if (typeof (myMovieClip [name]) == "movieclip"){
        trace("Mám dětský movie klip nazvaný" + name);
    }
}
```

Poznámka: Příkaz **for...in** iteruje vlastnosti objektů v prototypu řetězce iterovaného objektu. Pokud je prototyp dětského objektu rodič, bude **for...in** iterovat také vlastnosti rodiče.

Více informací o každé akci viz Kapitola 7, „ActionScript Slovník.“

Používání předdefinovaných funkcí

Funkce je blok kódu ActionScriptu, který může být znovu použit kdekoliv v animaci. Jestliže vložíte do funkce určité hodnoty (argumenty), funkce bude operovat a pracovat s těmito hodnotami. Funkce může také hodnoty vrátit. Flash má předdefinované funkce, které umožňují dosáhnout určitých informací a vykonat určité úkoly, jako je detekce kolizí (**hitTest**), získání hodnoty naposledy stisknuté klávesy (**keycode**) a získání čísla verze Flash Přehrávače hostícího animaci (**getVersion**).

Volání funkce

Funkce můžete volat v jakékoliv Časové ose z jakékoliv Časové osy, včetně natažené animace. Každá funkce má své vlastní charakteristiky a některé po vás požadují vložit určité hodnoty. Jestliže vložíte více argumentů než funkce požaduje, hodnoty navíc jsou ignorovány. Jestliže naopak nevložíte požadovaný argument, prázdné argumenty jsou určeny typem dat **undefined** (nedefinováno), který může způsobit chyby při exportu skriptu. Při volání funkce musí být funkce ve snímku dosaženém hrací hlavou.

Následuje seznam Flash předdefinovaných funkcí:

Boolean	localToGlobal
escape	maxscroll
eval	newline
false	number
getProperty	parseFloat
getTimer	parseInt
getVersion	random
globalToLocal	scroll
hitTest	String
int	targetPath
isFinite	true
isNaN	unescape
keycode	

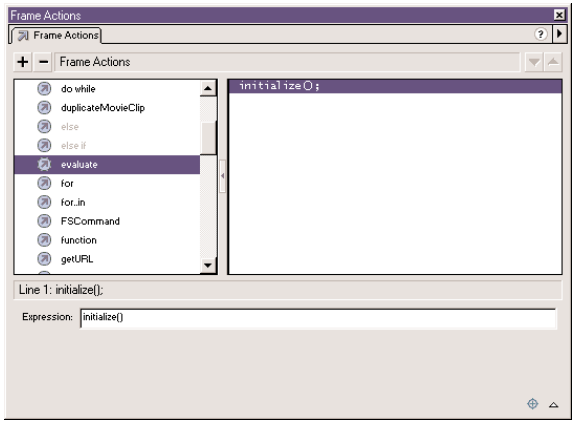
Poznámka: Řetězcové funkce nejsou doporučovány a proto nejsou zapsané ve výše uvedené tabulce.

Volání funkce v Expert Režimu:
Použijte jméno funkce a zadejte požadované argumenty dovnitř závorek. Následující příklad volá funkci **initialize**, která nepožaduje žádné argumenty:

```
initialize ();
```

Volání funkce v Normálním Režimu:

Použijte funkci **evalute**. Vložte jméno funkce a požadované argumenty do pole Expression.



Pro volání funkce na jiné Časové ose použijte cílovou cestu. Například, pro volání funkce **calculateTax**, která byla deklarována v instanci **functionsMovieClip**, použijte následující cestu:

```
_root.functionsMovieClip.calculateTax(total);
```

Poznámka: Uvnitř závorek zadávejte jakékoliv argumenty.

Více informací o každé funkci, včetně nedoporučených řetězcových funkcí viz Kapitola 7, „ActionScript Slovník.“

Vytváření speciálních funkcí

Můžete definovat funkce, aby na přiřazených hodnotách vykonávaly sérii příkazů. Vaše funkce také mohou vracet hodnoty. Jakmile je funkce definována, může být volána z jakékoliv Časové osy, včetně Časové osy nahrané animace.

Funkci si můžete představit jako "černou skříňku". Když je funkce volána, je jí přiřazen vstup (argumenty). Vykoná několik operací a pak generuje output (vrácenou hodnotu). Dobře napsané funkce má vhodně umístěné komentáře o svém vstupu, výstupu a účelu. Tak nemusí uživatel funkce přesně rozumět tomu, jak funkce funguje.

Definování funkce

Funkce, podobně jako proměnné, jsou připojeny k movie klipu, který je definuje. Když je funkce redefinována, nová definice nahradí starou.

Pro definování funkce použijte akci **function**, následovanou jménem funkce a zvolenými argumenty, které mají být do funkce vloženy, a příkazy ActionScriptu, které indikují co funkce dělá.

Následuje funkce pojmenovaná **Circle** s argumentem **radius**:

```
function Circle (radius) {
    this.radius = radius;
    this.area = Math.PI * radius * radius;
}
```

Poznámka: Klíčové slovo this použité v těle funkce, je odkaz na movie klip, ke kterému funkce patří.

Funkci můžete také definovat vytvořením **function literal** (doslovné funkce). Doslovná funkce je nepojmenovaná funkce, která je deklarována ve výrazu místo v příkazu. Doslovnou funkci můžete použít pro definování funkce, získání její hodnoty a její připojení k proměnné, v jednom výrazu:

```
area.(function() {return Math.PI * radius * radius;})(5);
```

Vkládání argumentů do funkce

Argumenty jsou prvky, na kterých funkce provádí svůj kód. (V této knize jsou zaměnitelné pojmy argument a parametr.) Například následující funkce má argumenty initials a finalScore:

```
function fillOutScorecard(initials, finalScore){
    scorecard.display = initials;
    scorecard.score = finalScore;
}
```

Při volání funkce musí být požadované argumenty do funkce vloženy. Funkce nahradí zadanými hodnotami argumenty v definici funkce. V tomto příkladě je **scorecard** jméno instance movie klipu; **display** a **score** jsou inputová textová pole v instanci. Následující volání funkce určuje proměnné **display** hodnotu **„JEB“** a proměnné **score** hodnotu **45000**:

```
fillOutScorecard(„JEB“,45000);
```

Argument **initials** ve funkci **fillOutScorecard** je podobný lokální proměnné; existuje, zatímco je funkce volána a přestává existovat, když funkce končí. Jestliže zapomenete na argumenty v průběhu volání funkce, jsou tyto zapomenuté argumenty vkládány jako **undefined** (nedefinované). Jestliže poskytnete argumenty navíc, které nejsou při volání funkce požadovány v deklaraci funkce, jsou tyto argumenty ignorovány.

Použití lokálních proměnných ve funkci

Lokální proměnné jsou cennými nástroji pro organizování kódu a usnadnění porozumění. Když funkce používá lokální proměnné, může své proměnné ukrýt před všemi ostatními skripty v animaci; lokální proměnné jsou platné pro tělo funkce a jsou zničeny, když funkce končí. Jakékoliv argumenty přiřazené k funkci budou také považovány za lokální proměnné.

Poznámka: Jestliže upravujete globální proměnné ve funkci, použijte pro dokumentaci těchto úprav skriptové komentáře.

Vracení hodnot z funkce

Pro vracení hodnot funkcí můžete použít akci **return**. Tato akce zastaví funkce a nahradí ji hodnotou akce **return**. Pokud Flash nenarazí na akci **return** před koncem funkce, je vrácen prázdný řetězec. Například následující funkce vrací mocninu argumentu **x**.

```
function sqr(x){
   return x * x;
}
```

Některé funkce vykonávají série úkolů bez vracení hodnoty. Například, následující funkce inicializuje sérii globálních proměnných:

```
function initialize (){
   boat_x=_root.boat._x;
   boat_y=_root.boat._y;
   car_x=_root.car._x;
   car_y=_root.car._y;
}
```

Volání funkce

Pro volání funkce pomocí panelu Akcí v Normálním Režimu použijte akci **evaluate**. Zadejte požadované argumenty uvnitř závorek. Funkci můžete volat na jakékoliv Časové ose z jakékoliv Časové osy, včetně natažené animace. Například následující příkaz volá funkce **sqr** v movie klipu **MathLib** na hlavní Časové ose, vkládá argument 3 a ukládá výsledek v proměnné **temp**:

```
var temp = _root.MathLib.sqr(3);
```

Ve Flash 4 můžete napsat skript pro volání funkce do snímků až za konec animace a volat ji jménem snímku pomocí akce **call**. Například, snímek nazvaný **initialize** a obsahující skript, byste mohli volat následovně:

```
call(„initialize");
```

Tento typ skriptu není skutečná funkce, protože nemůže vkládat argumenty a nemůže zadávat hodnotu. Ačkoliv akce **call** je stále funkční, ve Flash 5 její použití není doporučováno.

Používání předdefinovaných objektů

Pro dosažení a získání určitých druhů informací, můžete použít předdefinované objekty. Většina předdefinovaných objektů má **methods** (metody) (funkce připojené k objektu), které zadáváte pro obdržení hodnoty nebo k provedení akce. Například objekt Date poskytuje informaci ze systému hodin a objekt Sound umožňuje kontrolovat zvukové prvky v animaci.

Některé předdefinované objekty mají vlastnosti, jejichž hodnoty můžete číst. Například objekt Key, má konstantní hodnoty reprezentující klávesy na klávesnici. Každý objekt má své vlastní charakteristiky a schopnosti využitelné v animacích. Následuje seznam předdefinovaných objektů:

- **Array**
- **Boolean**
- **Color**
- **Date**
- **Key**
- **Math**
- **MovieClip**
- **Number**
- **Object**
- **Selection**
- **Sound**
- **String**
- **XML**
- **XMLSocket**

Instance movie klipů jsou v ActionScriptu prezentovány jako objekty. Metody předdefinovaných movie klipů můžete volat stejně, jako byste volali metody jakéhokoliv jiného ActionScript objektu.

Detailní informace o každém objektu viz Kapitola 7, „ActionScript Slovník."

Vytváření objektu

Existují dva způsoby pro vytvoření objektu: operátor **new** a inicializační operátor objektu (**{}**). Operátor **new** můžete použít pro vytvoření objektu z předdefinované třídy objektu nebo z uzpůsobené třídy objekt. Inicializační operátor objektu (**{}**) můžete použít pro vytvoření objektu obecného typu Objekt.

Při tvorbě objektu s použitím operátoru **new**, musí být použita konstrukční funkce. (Konstrukční funkce je jednoduše funkce, specifikující typ objektu, který má být vytvořen.) Předdefinované objekty ActionScriptu jsou v zásadě přepsané konstrukční funkce. Nový objekt **konkretizuje** nebo vytváří kopii objektu a přidává jí všechny vlastnosti a metody tohoto objektu. Toto je podobné jako přetažení movie klipu z Knihovny na Scénu. Například následující výrazy konkretizují objekt Date:

```
currentDate = new Date();
```

Metody některých předdefinovaných objektů lze volat i bez jejich konkretizací. Například následující výraz volá Math objekt, metodu **random**:

```
Math.random ();
```

Každý objekt požadující konstrukční funkci má odpovídající prvek v panelu nástrojů Akcí; například **new Color, new Date, new String** atd.

Vytvoření objektu pomocí operátoru `new` v Normálním Režimu:

1. Zvolte **setVariable**
2. Vložte identifikátor do pole Name.
3. Vložte **new Object, new Color** atd. do pole Value.
   Vložte argumenty požadované konstrukční funkcí v závorkách.
4. Zatrhněte box Expression pole Value.

Jestliže nezatrhnete box Expression, celá hodnota bude písmenný řetězec.

V následujícím kódu je objekt **c** vytvořen z konstruktoru Color:

```
c = new Color (this) ;
```

Poznámka: Jméno objektu je proměnná s typem dat připojených k ní.

Dosažení metody v Normálním Režimu:

1. Zvolte akci **evaluate**.
2. Vložte jméno objektu do pole Expression.
3. Vložte vlastnost objektu do pole Expression.

Použití operátoru inicializátor objektu ({}) v Normálním Režimu:

1. Zvolte akci **setVariable**.
2. Vložte jméno do pole Proměnná; toto je jméno nového objektu.
3. Vložte jméno, vlastnosti a páry hodnot oddělené dvojtečkami uvnitř operátoru inicializátor objektu (**{}**).

Například v tomto příkazu jsou jména vlastností **radius** a **area**, jejich hodnoty jsou 5 a hodnota výrazu:

```
myCircle = {radius:5, area:(PI * radius * radius)};
```

Výraz v závorkách bude ohodnocen a obdržená hodnota je hodnota proměnné **area**.

Pole a inicializátory objektu můžete „uhnízdit“, jako v tomto výrazu:

```
new Object = {name:"John Smith", projects:[„Flash","Dreamweaver"]};
```

Detailní informace o každém objektu viz Kapitola 7, „ActionScript Slovník.“

Připojení vlastností objektu

Pro připojení hodnoty vlastnosti v objektu použijte operátor tečka (**.**). Jméno objektu je na levé straně tečky a jméno vlastnosti na pravé straně. Například v následujícím skriptu je **myObject** objektem a **name** je vlastností:

```
myObject.name
```

Pro připojení hodnoty k vlastnosti v Normálním Režimu použijte akci setVariable:

```
myObject.name = „Allen";
```

Pro změnu hodnoty vlastnosti připojte novou hodnotu:

```
myObject.name = „Homer";
```

K připojení vlastností objektu můžete použít i operátor přístupové pole (**[]**). Viz „Tečkové operátory a operátory přístupové pole“.

Volání metody objektu

Metodu objektu můžete volat s použitím operátoru tečka, následovaného metodou. Například následující příklad volá metodu **setVolume** objektu Sound:

```
s = new Sound(this);
s.setVolume(50);
```

Pro volání metody předdefinovaného objektu v Normálním režimu použijte akci **evaluate**.

Použití objektu MovieClip

Ke kontrole instancí movie klipu symbolu na Scéně, můžete použít metody předdefinovaného objektu MovieClip,. Následující příklad spouští přehrávání instance **dateCounter**:

```
dateCounter.play ();
Detailní informace o objektu MovieClip viz Kapitola 7, „ActionScript
Slovník.“
```

Použití objektu Array

Objekt Array (Pole) je běžně používaný předdefinovaný ActionScript objekt, který uchovává, nebo obsahuje číslovaná data. Každý číslovaný údaj se nazývá prvek pole a číslo přiřazené prvku se nazývá index. To je užitečné pro ukládání a znovuzískání určitých typů informací, jako jsou seznamy nebo sekvence pohybu ve hře.

Prvky objektu Array můžete připojit stejně, jako byste připojovali vlastnost jakéhokoliv objektu:

```
move[0] ==„a2a4“;
move[1] == „h7h5“;
move[2] == „b1c3“;
...
move[100] ==„e3e4“;
```

Při volání druhého prvku pole použijte výraz **move[1]**.

Objekt Array má předdefinovanou vlastnost **length**, což je hodnota počtu prvků v poli. Když je prvek objektu Array připojen a index prvku je pozitivní celé číslo jako **index >= length**, length je automaticky aktualizován na **index + 1**.

Použití speciálních objektů

Pro organizaci informací ve skriptech, pro jejich snadnější skladování a přístup k nim, můžete vytvořit speciální objekty pomocí definování vlastností a metod objektu. Poté, co vytvoříte hlavní objekt nebo „třídu“, můžete použít nebo „konkretizovat“ kopie (tzn. Instance) tohoto objektu. Toto vám umožní znovu použít kód a neměnit velikost souboru.

Objekt je komplexní typ dat obsahující žádnou nebo více vlastností. Každá vlastnost, jako proměnná, má jméno a hodnotu. Vlastnosti jsou připojeny k objektu a obsahují hodnoty, které mohou být měněny a znovu získávány. Tyto hodnoty mohou být jakéhokoliv typu dat: řetězec, číslo, Booleánské, objekt, movie klip nebo nedefinované. Následující vlastnosti jsou různé typy dat:

```
customer.name = "Jane Doe";
customer.age = 30;
customer.member = true;
customer.account.currentRecord = 000609;
customer.mcInstanceName._visible = true;
```

Vlastnost objektu může být také objekt. Ve 4. Řádku předchozího příkladu je **account** vlastnost objektu **customer** a **currentRecord** je vlastnost objektu **account**. Typ dat vlastnosti **currentRecord** je číslo.

Vytváření objektu

Pro vytvoření objektu z konstrukční funkce, můžete použít operátor **new**. Konstrukční funkci je vždy dáváno stejné jméno jako je typ objektu, který vytváří. Například konstruktor, který vytváří objekt **account** by byl nazván **Account**. Následující výraz vytváří nový objekt z funkce nazvané **MyConstructorFunction**:

```
new MyConstructorFunction (argument1,argument2,...argumentN);
```

Když je volána **MyConstructorFunction**, Flash propustí schovaný argument **this**, který je odkazem na objekt vytvářející **MyConstructorFunction**. Při definici konstruktoru, **this** umožní odkazovat na objekty, které konstruktor vytvoří. Například následující je konstrukční funkce, která vytváří kruh:

```
function Circle(radius){
    this. radius = radius;
    this.area = Math.PI * radius * radius;
}
```

Konstruktory funkcí jsou běžně používány jako zadání metody objektu.

```
function Area(){
    this.circleArea = Math.PI * radius * radius;
}
```

Objekt použitý ve skriptu musí být připojen k proměnné. Pro vytvoření objektu nový kruh s poloměrem 5 použijte operátor **new** a připojte ho k lokální proměnné **myCircle**:

```
var myCircle = new Circle(5);
```

Poznámka: Objekty mají stejné umístění jako proměnné, ke kterým jsou připojeny. Viz „Umísťování proměnné“.

Vytváření dědičnosti

Všechny funkce mají vlastnost **prototype**, která je vytvořena automaticky při definování funkce. Jestliže používáte konstrukční funkci pro vytvoření nového objektu, všechny vlastnosti a metody konstrukční vlastnosti **prototype** se stávají vlastnostmi a metodami vlastnosti **_proto_** nového objektu. Vlastnost **prototype** indikuje nastavené hodnoty vlastnosti pro objekty vytvořené touto funkcí. Předávání hodnot použitím vlastností **_proto_** a **prototype** je nazváno dědičnost.

Dědičnost postupuje podle definitivní hierarchie. Když voláte vlastnost nebo metodu objektu, ActionScript hledá v objektu, zda takový prvek existuje. Jestliže neexistuje, ActionScript hledá ve vlastnosti objektu **_proto_** informaci(**object._proto_**). Jestliže volaná vlastnost není vlastností objektu **_proto_**, ActionScript hledá v **object._proto_._proto_**.

Je běžné připojovat metody k objektu jejich určením do vlastnosti objektu **prototype**. Následující kroky popisují, jak definici provést:

1. Definujte konstrukční funkci **Circle**:

```
function Circle(radius){
    this. radius = radius;
}
```

2. Definujte metodu **area** objektu **Circle**. Metoda **area** bude počítat obsah kruhu. Pro definování metody **area** můžete použít doslovnou funkci a nastavit vlastnost oblasti objektu prototypu kruhu následovně:

```
Circle.prototype.area = function(){
    return Math.PI * this.radius * this.radius;
  }
```

3. Vytvořte instanci objektu **Circle**:

```
var myCircle = new Circle(4);
```

4. Volejte metodu **area** nového objektu **myCircle**:

```
var myCircleArea = myCircle.area ()
```


ActionScript hledá objekt **myCircle**, metodu **area**. Protože objekt nemá metodu **area**, v jeho prototypu objektu **Circle.prototype** je hledána metoda **area**. ActionScript ji najde a volá ji.

Metodu také můžete připojit k objektu jejím připojením ke každé jednotlivé instanci objektu:

```
function Circle(radius){
    this.radius = radius;
    this.area = function(){
        return Math.PI * this.radius * this.radius;
    }
}
```

Tato technika ale není doporučována. Použití objektu **prototype** je účinnější, protože je nezbytná pouze jedna definice **area** a tato definice je automaticky kopírována do všech instancí vytvořených funkcí **Circle**.

Vlastnost **prototype** je podporována verzí Flash Přehrávač 5 a novější. Více informací viz Kapitola 7, „ActionScript Slovník.“

Otevírání Flash 4 souborů

ActionScript se značně změnil od uveřejnění Flash 5. Nyní je to objektově orientovaný jazyk s několika typy dat a dot syntaxí. Flash 4 ActionScript měl pouze jeden typ dat: řetězec. Používal různé typy operátorů ve výrazech pro indikaci, zda by s hodnotou mělo být zacházeno jako s řetězcem nebo jako s číslem. Ve Flash 5 můžete použít jednu sadu operátorů na všechny typy dat.

Když používáte Flash 5 pro otevírání souboru, který byl vytvořen ve Flash 4, Flash automaticky konvertuje ActonScript výrazy tak, aby je učinil kompatibilní s novou syntaxí Flash 5. Následující konverze typů dat a operátorů vidíte ve vašem ActionScript kódu:

- Operátor = ve Flash 4 byl používán pro numerickou rovnost. Ve Flash 5 je == operátor rovnosti a = je operátor určení. Jakékoliv operátory = ve Flash 4 souborech jsou automaticky konvertovány na ==.

- Flash automaticky provádí konverze pro ujištění, že se operátory chovají korektně. Kvůli zavedení mnoha typů dat mají následující operátory nové významy:
  +, ==, !=, <>, <, >, >=, <=

- Ve Flash 4 byly tyto operátory vždy numerickými operátory. Ve Flash 5 se chovají různě v závislosti na typu dat operandů. Aby se zabránilo sémantickým odlišnostem v importovaných souborech, funkce **Number** je vložena do všech operandů k těmto operátorům. (Konstantní čísla jsou vždy jasná, proto nejsou vkládána do **Number**).

- Ve Flash 4 úniková sekvence \n generovaná carriage return znakem (ASCII 13). Ve Flash 5 pro shodu se standardem ECMA-262, \n generuje line-feed znak (ASCII 10). Sekvence \n ve Flash 4 FLA souborech je automaticky konvertována na \r.

- Operátor & ve Flash 4 byl používán pro přidání řetězce. Ve Flash 5 je **&** bitový AND operátor. Operátor přidání řetězce je nyní nazván **add**. Jakékoliv operátory **&** ve Flash 4 souborech jsou automaticky konvertovány na operátory **add**.

- Mnoho funkcí ve Flash 4 nepožaduje zavření závorek, například **Get Timer**, **Set Variable**, **Stop** a **Play**. Ve Flash 5 funkce **getTimer** a všechny ostatní funkce nyní zavřené závorky požadují. Tyto závorky jsou automaticky přidávány během konverze.

- Když je ve Flash 5 funkce **getProperty** vykonána na movie klipu který neexistuje, vrací hodnotu **undefined**, ne 0. Ve Flash 5 ActionScriptu **undefined ==** 0 je **false**. Flash fixuje tento problém při konverzi na Flash 4 soubory, zavedením funkcí **Number** v rovnicových srovnáních. V následujícím příkladu **Number** přinutí **undefined** aby bylo konvertováno na 0, aby bylo tak dosaženo srovnání:

```
getProperty("clip", _width) == 0
Number(getProperty("clip", _width)) == Number(0)
```

Poznámka: Jestliže jste použili ve skriptu napsaném ve Flash 4 jakákoliv klíčová slova rezervovaná Flashem 5 jako jména proměnných, syntaxe ve Flash 5 nahlásí chybu. Toto odstraníte přejmenováním všech takových proměnných ve skriptu. Viz „Klíčová slova“.

Použití Flash 5 k vytvoření obsahu pro Flash 4

Jestliže používáte Flash 5 k vytvoření obsahu pro Flash 4 Přehrávač (exportováním jako Flash 4), nebudete moci využít všechny nové vlastnosti Flash 5 ActionScriptu. Nicméně, mnoho jich zůstává stále dostupných. Flash 4 ActionScript má pouze jeden základní typ dat, který je používán pro číselnou i řetězcovou manipulaci. Když autorizujete movie pro Flash 4 Přehrávač, musíte použít nežádoucí řetězcové operátory umístěné v kategorii String Operators v boxu nástrojů.

Následující rysy Flash 5 můžete použít při exportu do .swf souboru formátu Flash:

- Operátor pole a přístupové pole (**[]**).

- Tečkový operátor (**.**).

- Logické operátory, určovací operátory a před-zvyšovací a po-zvyšovací/snižovací operátory.

- Modulo operátor (**%**), všechny metody a vlastnosti objektu Math. Tyto operátory a funkce nejsou původně podporovány Flash 4 Přehrávačem. Flash 5 je musí exportovat jako série přibližných hodnot. To znamená, že výsledky jsou pouze přibližné. Navíc, vzhledem k započtení sérií přibližností do SWF souboru, tyto funkce zabírají více prostoru ve Flash 4 SWF souborech než ve Flash 5 SWF souborech.

- Akce **for, while, do**...**while**, **break** a **continue**.

- Akce **print** a **printAsBitmap**.

Následující rysy Flash 5 nemůžete použít při exportu do **.swf** souboru formátu Flash:

- Speciální funkce
- XML podpora
- Lokální proměnné
- Předdefinované objekty (kromě Math)
- Movie clip akce
- Typy dat Multiple
- **eval** s dot syntaxí (například, **eval("_root.movieclip.variable"))**
- **return**
- **new**
- **delete**
- **typeof**
- **for..in**
- **keycode**
- **targetPath**
- **escape**
- **globalToLocal a localToGlobal**
- **hitTest**
- **isFinite a inNaN**
- **parseFloat a parseInt**
- **unescape**
- **_xmouse** a **_ymouse**
- **_quality**

KAPITOLA 3

Vytváření Interakce s ActionScriptem

Interaktivní animace uživatele zahrnuje do děje a také s nimi počítá. Použitím klávesnice, myši nebo obojího mohou uživatelé skákat do různých částí animace, hýbat objekty, vkládat informace (data, údaje), klikat na tlačítka a provádět mnoho dalších interaktivních operací. Interaktivní animace vytváříte nastavením skriptů, které běží, když se objeví určité události. Události mohou spustit skript pokud hrací hlava dosáhne určitého snímku, nebo když se natáhne nebo stáhne ze scény movie klip, nebo když uživatel klikne na tlačítko nebo stlačí klávesy na klávesnici. ActionScript lze použít k vytváření skriptů, které říkají jaká akce se má po dané události spustit.

Následujícími způsoby může uživatel interaktivně ovládat animaci:

- Hraní a zastavování animace
- Upravování kvality zobrazení animace
- Zastavení všech zvuků
- Skok na snímek nebo scénu
- Skok na jinou URL
- Kontrola, zda je snímek natažen
- Natažení a stažení dodatečných animací (movies)

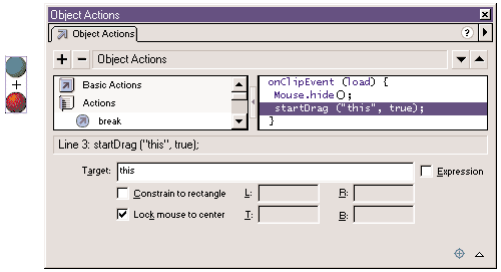
Detailní informace o těchto akcích, viz **Using Flash**.

Pro vytváření složitější uživatelské interaktivity musíte pochopit následující techniky:

- Vytváření speciálního kursoru
- Získání pozice myši
- Ovládnutí stisků kláves
- Vytváření rolujícího textového pole
- Nastavení hodnot barev
- Vytváření kontrol zvuku
- Detekce kolizí

Vytváření speciálního kursoru

Schování standardního kursoru (prezentace ukazatele myši na obrazovce) provedete metodou **hide** předdefinovaného objektu Mouse. Při použití movie klipu jako speciálního kursoru použijete akci **startDrag**.



Vytvoření speciálního kursoru:

1. Vytvořte movie klip, který použijete jako speciální kursor.
2. Zvolte movie klip instanci na Scéně.
3. Zvolte Window > Action pro otevření panelu Object Actions.
4. V seznamu Toolbox list zvolte Objects, potom zvolte Mouse a přetáhněte **hide** do okna Scénář.
   Kód by měl vypadat takto:

```
onClipEvent(load){
    Mouse.hide();
}
```

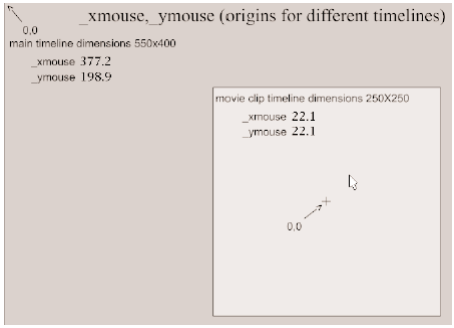
5. V seznamu Toolbox list zvolte Actions; potom přetáhněte **startDrag** do okna Scénář.
6. Zvolte LockMouse to center a zatrhněte Expressions.
   Kód by měl vypadat takto:

```
onClipEvent(load){
    Mouse.hide()
startDrag(this, true);
}
```

7. Zvolte Control > Test Movie pro zkoušku použití speciálního kursoru.
   Tlačítka budou stále funkční i pokud použijete speciální kursor. Je dobré dát speciální kursor do vrchní vrstvy Časové osy, aby se myš pohybovala nad tlačítky a dalšími objekty.
   Více informací o způsobech objektu Myš viz Kapitola 7, „ActionScript Slovník.“

Získání pozice myši

Vlastnosti **_xmouse** a **_ymouse** můžete použít k nalezení umístění ukazatele myši (kursoru) v animaci. Každá Časová osa má vlastnost **_xmouse** a **_ymouse**, která udává umístění myši uvnitř systému souřadnic.



Následující příkaz by mohl být umístěn na jakékoliv Časové ose animace, na úrovni **_level0**, pro získání **_xmouse** pozice uvnitř hlavní Časové osy:

```
x_pos = _root._xmouse ;
```

Pro určení pozice myši uvnitř movie klipu lze použít jméno instance movie klipu. Například následující příkaz by mohl být umístěn na jakékoliv Časové ose animace, v úrovni **_level0**, pro získání **_ymouse** pozice v instanci **myMovieClip**:

```
y_pos = _root.myMovieClip._ymouse
```

Pozici myši uvnitř movie klipu, můžete také určit použitím vlastností **_xmouse** a **_ymouse** v klip akci:

```
onClipEvent(enterFrame){
    xmousePosition = _xmouse;
    ymousePosition = _ymouse;
}
```

Proměnné **x_pos** a **y_pos** jsou použity jako schránky obsahující hodnoty pozicí myši. Tyto proměnné můžete použít v jakémkoliv skriptu v animaci. V následujícím příkladě se hodnoty **x_pos** a **y_pos** updatují pokaždé, když uživatel pohne s myší.

```
onClipEvent(mouseMove){
    x_pos = _root._xmouse;
    y_pos = _root._ymouse;
}
```

Více informací o vlastnostech **_xmouse** a **_ymouse** viz Kapitola Z, „ActionScript slovník.“

Ovládnutí stisků kláves

Metody předdefinovaného objektu Key, můžete použít pro určení poslední klávesy, kterou uživatel stiskl. Objekt Key nepožaduje konstrukční funkci; pro použití těchto metod jednoduše voláte samotný objekt:

```
Key.getCode();
```

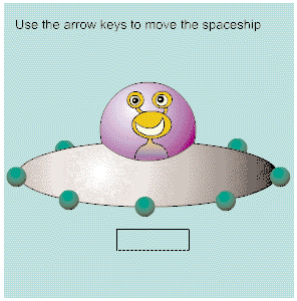
Můžete získat jak virtuální kódy kláves, tak ASCII hodnoty stisknutí kláves:

- Pro získání virtuálního kódu klávesy, která byla naposledy stisknuta použijte metodu **getCode**.
- Pro získání ASCII hodnoty naposledy stisknuté klávesy použijte metodu **getAscii**.

Virtuální kód klávesy je stanoven pro každou fyzickou klávesu na klávesnici. Například klávesa levá šipka má virtuální kód 37. Použitím virtuálního kódu klávesy zajistíte, že vámi určená klávesa bude na klávesnici, bez ohledu na jazyk nebo platformu, opravdu ta kterou jste určili.

ASCII (American Standard Code for Information Interchange) hodnoty jsou určeny k prvním 127 znakům v každé sadě znaků. ASCII hodnoty poskytují informaci o znaku na obrazovce. Například písmeno „A“ a písmeno „a“ mají různé ASCII hodnoty.

Běžné místo pro použití **Key.getCode** je ovladač **onClipEvent**. Vložení **keyDown** jako parametru, instruuje ovladač, aby zkontroloval hodnotu naposledy stisknuté klávesy pouze tehdy, je-li klávesa momentálně stisknuta. Tento příklad používá **Key.getCode** v příkazu **if**, pro vytvoření navigačních kontrol pro kosmickou loď.



Vytvoření klávesových ovladačů v animaci:

1. Rozhodněte se, které klávesy použijete a určete jejich virtuální klávesové kódy použitím jedním z těchto postupů:
   - Podívejte se do seznamu kódu kláves v Dodatku B, „Klávesy na Klávesnici a Hodnoty Kódů Kláves.“
   - Použijte konstantu Key object. (V seznamu Toolbox list zvolte Objects, potom zvolte Key. Konstanty jsou zapsány velkými písmeny.)
   - Zadejte následující akci klipu, potom zvolte Control > Test Movie a stiskněte požadovanou klávesu:

```
onClipEvent(keyDown){
        trace(Key.getCode());
}
```

2. Zvolte movie klip na Scéně.
3. Zvolte Window > Action.
4. Dvakrát stiskněte akci **onClipEvent** v kategorii Actions, v boxu nástrojů.
5. Zvolte událost **keyDown** v panelu parametry.
6. Dvakrát klikněte na akci **if** v kategorii Actions, v boxu nástrojů.
7. Klikněte na parametr Condition, zvolte Objects; potom zvolte Key a **getCode**.
8. Dvakrát klikněte na operátor rovnosti (==) v kategorii Operátory, v boxu nástrojů.
9. Vložte virtuální kód klávesy na pravou stranu operátoru rovnosti.

Váš kód by měl vypadat takto:

```
onClipEvent(keyDown){
    if (Key.getCode()==32){
    }
}
```


10. Zvolte akci, která má být provedena po stisknutí správné klávesy.

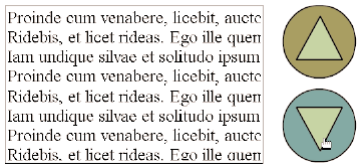
Například následující akce způsobí, že hlavní Časová osa půjde na další snímek, pokud je stisknut Mezerník (32):

```
onClipEvent(keyDown){
    if (Key.getCode() == 32){
        nextFrame();
    }
}
```

Více informací o způsobech objektu Klávesa viz Kapitola 7, „ActionScript Slovník.“

Vytváření rolujícího textového pole

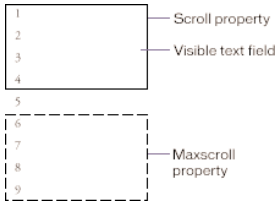
Pro vytvoření rolujícího textového pole použijte vlastnosti **scroll** a **maxscroll**.



V panelu Text Options zadejte jméno proměnné v jakémkoliv nastavení textového pole - Input Text nebo Dynamic Text. Textové pole se bude chovat jako okno zobrazující hodnotu této proměnné.

Každá proměnná spojená s textovým polem má vlastnost **scroll** a **maxscroll**. Tyto vlastnosti můžete využít při rolování textu v textovém poli. Vlastnost scroll udává číslo nejvýše viditelného řádku v textovém poli; tuto vlastnost můžete nastavit a znovu získat. Vlastnost **maxscroll** udává nejvýše viditelný řádek v textovém poli, když je vidět spodní řádek textu; tuto vlastnost můžete číst, ale ne nastavit.

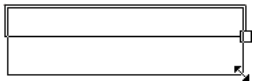
Například předpokládejme, že máte textové pole dlouhé čtyři řádky. Jestliže obsahuje proměnnou **speech**, která by zaplnila devět řádků textového pole, tak zobrazena najednou (identifikována pevným boxem) může být pouze část proměnné **speech**:



Získat tyto vlastnosti můžete dot syntaxe:
`textFieldVariable.scroll`
`myMovieClip.textFieldVariable.scroll`
`textFieldVariable.maxscroll`
`myMovieClip.textFieldVariable.maxscroll`

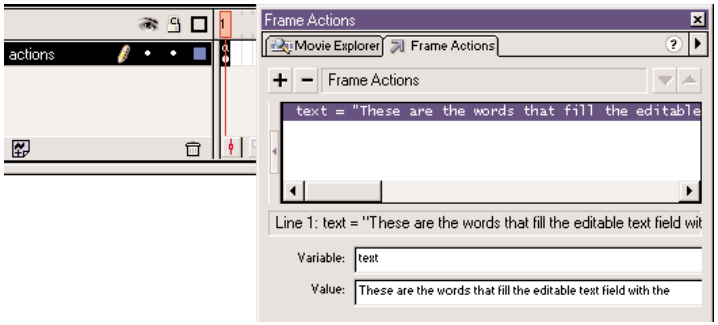
Vytvoření rolujícího textového pole:

1. Přetáhněte textové pole na Scénu.
2. Zvolte Window > Panels > Text Options.
3. Zvolte Input Text z pop-up menu.
4. Vložte jméno textové proměnné do pole Variable.
5. Táhněte pravý dolní roh textového pole pro změnu velikosti textového pole.



6. Zvolte Window > Action.
7. Zvolte snímek 1 na hlavní Časové ose a určete akci **`setVariable`**, která nastavuje hodnotu textu.

V poli se žádný text neobjeví dokud není nastavena proměnná. Proto, ačkoliv můžete tuto akci připojit k jakémukoliv snímku, tlačítku nebo movie klipu, je dobré ji přiřadit k snímku 1 na hlavní Časové ose:

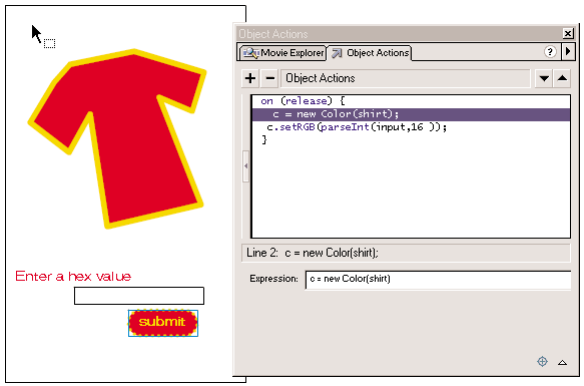


8. Zvolte Window > Common Libraries > Buttons, a přetáhněte tlačítko na Scénu.
9. Stiskněte Alt (Windows) nebo Option (Macintosh) a táhněte tlačítko pro vytvoření kopie.
10. Zvolte vrchní tlačítko a zvolte Window > Action.
11. Přetáhněte akci **`setVariable`** z boxu nástrojů do okna Skript v panelu Akcí.
12. Vložte **text.scroll** do boxu Variable.
13. Vložte **text.scroll -1** do boxu Value a zatrhněte Expression.
14. Zvolte tlačítko Šipka Dolů a určete následující akci **`setVariable`**:
    `text.scroll = text.scroll + 1;`
15. Zvolte Control > Test Movie pro otestování rolování textového pole.

Více informací o vlastnostech **`scroll`** a **`maxscroll`** viz Kapitola 7, „ActionScript Slovník.“

Nastavení hodnot barev

Metody předdefinovaného objektu Color, můžete použít pro upravení barvy movie klipu. Metoda **`setRGB`** určuje objektu hexadecimální RGB (red-červená, green-zelená, blue-modrá) hodnoty a metoda **`setTransform`** nastavuje procento a vyrovnávací hodnoty pro červenou, zelenou, modrou a průhledné (alfa) komponenty barvy. Následující příklad používá **`setRGB`** pro změnu barvy objektu založenou na vkladu uživatele.



Pro použití objektu Color musíte vytvořit instanci objektu a použít ji na movie klip.

Nastavení hodnoty barvy movie klipu:

1. Zvolte movie klip na Scéně a zvolte Window > Panels > Instance.
2. Vložte jméno instance **colorTarget** do boxu Name.
3. Přetáhněte textové pole na Scénu.
4. Zvolte Window > Panels > Text Option a určete jméno proměnné **`input`**.
5. Přetáhněte tlačítko na Scénu a označte (zvolte) ho.
6. Zvolte Window > Actions.
7. Přetáhněte akci **`setVariable`** z boxu nástrojů do okna Skript.
8. Do boxu Variable vložte **`c`**.
9. V boxu nástrojů zvolte Objects, potom Color a přetáhněte **`new Color`** do boxu Value.
10. Zatrhněte Expression.
11. Klikněte na tlačítko Target Path a zvolte **`colorTarget`**. Klikněte OK.

Kód v okně Scénář by měl vypadat takto:

```
on(release){
    c = new Color(colorTarget);
}
```

12. Přetáhněte akci **`evaluate`** z boxu nástrojů do okna Skript.
13. Vložte **`c`** do boxu Expression.
14. V kategorii Objects seznamu Toolbox list zvolte Color; potom přetáhněte **`setRGB`** do boxu Expression.
15. Zvolte Functions a přetáhněte parseInt do boxu Expression.

Kód by měl vypadat takto:

```
on(release){
    c = new Color(colorTarget);
    c.setRGB(parseInt(string, radix));
}
```


16. Pro řetězcový argument **`parseInt`** vložte **input**.

Řetězec, který má být analyzován je hodnota vložená do editovatelného textového pole.

17. Pro radix argument **`parseInt`** vložte **16**.

Radix je základ číselného systému, který má být analyzován. V tomto případě, je 16 základem hexadecimálního systému, který používá objekt Color. Kód by měl vypadat takto:

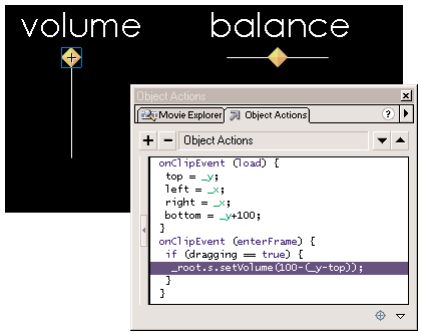
```
on(release){
    c = new Color(colorTarget);
    c.setRGB(parseInt(input, 16));
}
```

18. Zvolte Control > Test Movie pro změnu barvy movie klipu.

Více informací o způsobech objektu Color viz Kapitola 7, „ActionScript Slovník.“

Vytváření ovladačů zvuku

Pro kontrolu a ovládání zvuků v animaci používáte předdefinovaný objekt Sound. Pro použití metod objektu Sound musíte nejprve vytvořit nový objekt Sound. Potom můžete, zatímco animace běží, použít pro vložení zvuku z knihovny metodu **`attachSound`**. Metoda **`setVolume`** kontroluje hlasitost a **`setPan`** upravuje pravou a levou rovnováhu zvuku.



Připojení zvuku k Časové ose:

1. Zvolte File > Import pro importování zvuku.
2. Zvolte zvuk v knihovně a zvolte Linkage z menu Options.
3. Zvolte Export This Symbol a dejte mu identifikátor **mySound**.
4. Zvolte snímek 1 na hlavní Časové ose a zvolte Window > Actions.
5. Přetáhněte akci **`setVariable`** z boxu nástrojů do okna Skript.
6. Vložte s do boxu Value.
7. V seznamu Toolbox list zvolte Objects, potom zvolte Sound a přetáhněte **`new Sound`** do boxu Value.

Kód by měl vypadat takto:

```
s = new Sound ();
```

8. Dvakrát klikněte na akci **`evaluate`** v boxu nástrojů.
9. Vložte s do boxu Expression.
10. V kategorii Objects seznamu Toolbox list zvolte Sound, potom přetáhněte **`attachSound`** do boxu Expression.
11. Vložte „**mySound**“ do ID argumentu **`attachSound`**.
12. Dvakrát klikněte na akci **`evaluate`** v boxu nástrojů.
13. Vložte s do boxu Expression.
14. V kategorii Objects zvolte Sound, potom přetáhněte **`start`** do boxu Expression.

Kód by měl vypadat takto:

```
s = new Sound();
s.attachSound("mySound");
s.start();
```

15. Zvolte Control > Test Movie, abyste slyšeli přehrát zvuk.

Vytvoření posuvného ovladače hlasitosti:

1. Táhněte tlačítko na Scénu.
2. Zvolte tlačítko a zvolte Insert > Convert to Symbol. Zvolte behavior movie clip.

Toto vytvoří movie klip s tlačítkem na jeho prvním snímku.

3. Zvolte movie klip a zvolte Edit > Edit Symbol.
4. Zvolte tlačítko a Window > Action.
5. Vložte následující akce:

```
on (press){
    startDrag ("", false, left, top, right, bottom);
    dragging = true;
}
on (release, releaseOutside){
    stopDrag ();
    dragging = false;
}
```

Parametry **`startDrag`** - levý, horní, pravý a spodní jsou proměnné nastavené v akci klipu.

6. Zvolte Edit > Edit Movie pro návrat na hlavní Časovou osu.
7. Zvolte movie klip na Scéně.
8. Vložte následující akce:

```
onClipEvent (load){
    top = _y;
    left = _x;
    right = _x;
    bottom = _y + 100;
}
onClipEvent(enterFrame){
    if (dragging == true){
          _root.s.setVolume(100 - (_y - top));
    }
}
```

9. Zvolte Control > Test Movie pro použití ovladače hlasitosti.

Vytvoření pohyblivého ovladače vyrovnávání:

1. Přetáhněte tlačítko na Scénu.

2. Zvolte tlačítko a zvolte Insert > Convert to Symbol. Zvolte vlastnost movie klipu.

3. Zvolte movie klip a zvolte Edit > Edit Symbol.

4. Zvolte tlačítko a Window > Actions.

5. Vložte následující akce:

```
on (press){
    startDrag ("", false, left, top, right, bottom);
    dragging = true;
}
on (release, releaseOutside){
    stopDrag ();
    dragging = false;
}
```

Parametry **startDrag** - levý, horní, pravý a spodní jsou proměnné nastavené v akci klipu.

6. Zvolte Edit > Edit Movie pro návrat na hlavní Časovou osu.

7. Zvolte movie klip na Scéně.

8. Vložte následující akce:

```
onClipEvent(load){
    top = _y;
    bottom = _y;
    left = _x - 50;
    right = _x + 50;
    center = _x;
}
onClipEvent(enterFrame){
    if (dragging == true){
        _root.s.setPan((_x - center) * 2);
    }
}
```

9. Zvolte Control > Test Movie pro použití pohyblivého vyrovnávače.

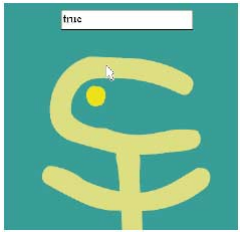
Více informací o metodách objektu Sound viz Kapitola 7, „ActionScript Slovník.“

Detekce kolizí

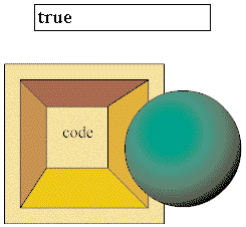
Metodu **hitTest** objektu MovieClip, můžete použít pro detekci kolizí v animaci. Metoda **hitTest** kontroluje, zda objekt koliduje s movie klipem a dává Booleánskou hodnotu (**true** nebo **false**). Parametry metody **hitTest** můžete použít pro specifikování souřadnic **x** a **y** oblasti na Scéně, nebo použít cílovou cestu jiného movie klipu jako hit (zasahované) oblasti. Každý movie klip v animaci je instance MovieClip objektu. Toto umožňuje volat metody objektu z jakékoliv instance:

```
myMovieClip.hitTest (target) ;
```

Metodu **hitTest** můžete použít pro testování kolize v movie klipu a jednotlivého bodu.



Také můžete metodu **hitTest** použít pro testování kolize mezi dvěma movie klipy.



Provedení detekce kolize mezi movie klipem a bodem na Scéně:

1. Zvolte movie klip na Scéně.

2. Zvolte Window > Actions pro otevření panelu Objekt Akce.

3. Dvakrát klikněte na **trace** v kategorii Actions, v boxu nástrojů.

4. Zatrhněte Expression a vložte následující Výraz:

```
trace (this.hitTest(_root._xmouse, _root._ymouse, true);
```

Tento příklad používá vlastnosti **_xmouse** a **_ymouse** jako souřadnice **x** a **y** pro hit (zasahovanou) oblast a posílá výsledky do okna Output v Režimu Test-Movie. Pro zobrazení výsledků také můžete nastavit textové pole na Scéně, nebo použít výsledky v příkazu **if**.

5. Zvolte Control > Test Movie a posuňte myš na movie klip pro testování kolize.

Provedení detekce kolize na dvou movie klipech:

1. Přetáhněte dva movie klipy na Scénu a dejte jim jména instancí **mcHitArea** a **mcDrag**.
2. Vytvořte textové pole na Scéně a vložte **status** do Variable v boxu Text Options.
3. Zvolte **`mcHitArea`** a Window > Actions.
4. Dvakrát klikněte na **`evaluate`** v boxu nástrojů.
5. Vložte následující kód do boxu Expression vybráním položek z boxu nástrojů:
   ```
   _root.status = this.hitTest(_root.mcDrag);
   ```
6. Zvolte akci **`onClipEvent`** v okně Skript a zvolte **`enterFrame`** jako událost.
7. Zvolte **`mcDrag`** a Window > Actions.
8. Dvakrát klikněte na **`startDrag`** v boxu nástrojů.
9. Zvolte Lock Mouse to Center.
10. Zvolte akci **`onClipEvent`** v okně Skript a zvolte událost **`Mouse down`**.
11. Dvakrát klikněte na **`stopDrag`** v boxu nástrojů.
12. Zvolte akci **`onClipEvent`** v okně Skript a zvolte událost **`Mouse up`**.
13. Zvolte Control > Test Movie a táhněte movie klip pro testování detekce kolize.

Více informací o metodách hitTest viz Kapitola 7, „ActionScript Slovník.“

KAPITOLA 4

Práce s Movie Klipy

Movie klip je mini animace, která má svoji vlastní Časovou osu a vlastnosti. Symbol movie klip umístěný v Knihovně, může být v animaci použit několikrát; každé použití je nazváno instance movie klipu. Movie klipy mohou být umístěny uvnitř sebe navzájem. Pro odlišení instancí používejte pro každou instanci jedinečné jméno.

Na Časové ose movie klipu může být umístěn jakýkoliv objekt, včetně jiných movie klipů. Animace, které jsou nataženy do Flash Přehrávače pomocí **`loadMovie`** jsou také mini Flash animace. Každý movie klip, natažená animace a hlavní Časová osa v animaci jsou objekty s vlastnostmi a metodami, se kterými může být manipulováno pomocí ActionScriptu k vytváření komplexní, nelineární a interaktivní animace.

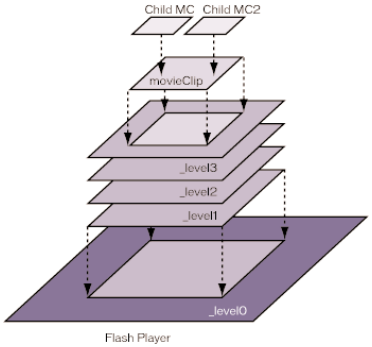
Movie klipy kontrolujete pomocí akcí a metod MovieClip objektů. V movie klipu jsou akce a metody připojeny buď, ke snímkům nebo k tlačítkům (akce snímků a tlačítek), nebo ke specifické movie klip instanci (akce klipu). Akce v movie klipu mohou kontrolovat jakoukoliv Časovou osu v animaci. Časovou osu můžete ovládat pomocí použití cílové cesty (target path). Cílová cesta udává umístění Časové osy v animaci.

Z movie klipu můžete vytvořit „smart“ klip; movie klip s ActionScriptem, který může být přeprogramován bez použití panelu Akcí. Smart klipy činí jednodušší komunikaci mezi programátorem a návrhářem.

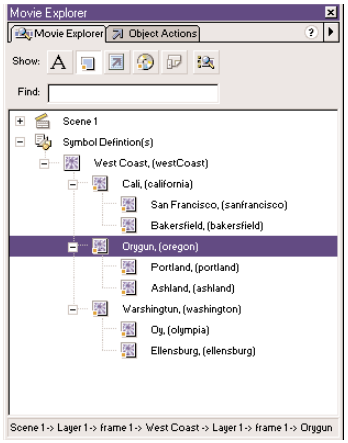
O několika Časových osách

Každá Flash animace má ve Flash Přehrávači hlavní Časovou osu umístěnou v úrovni 0 (level0). Pro natažení dalších Flash animací (.swf souborů) do Flash Přehrávače na jakoukoliv úroveň vyšší než 0 (například úroveň1, úroveň 2, úroveň 15), můžete použít akci **`loadMovie`**. Každá animace natažená do nějaké úrovně má svou Časovou osu.

Flash animace na jakékoliv úrovni, mohou mít na svých Časových osách instance movie klipu. Každá instance movie klipu má také Časovou osu a může obsahovat další movie klipy, které mají také svoje Časové osy. Časové osy movie klipů a úrovně ve Flash Přehrávači jsou hierarchicky organizovány tak, že objekty v animaci můžete organizovat a kontrolovat velmi snadno.



Ve Flash je tato hierarchie úrovní a movie klipů nazvána **display list** (seznam zobrazení) a je dispozici k nahlédnutí a úpravám v Movie Exploreru. Do seznamu zobrazení můžete nahlédnout také v Debuggeru, pokud si přehrajete animaci v Režimu Test-Movie, samostatném Flash Přehrávači nebo na Webovém browseru.



Časové osy Flash animací jsou objekty a všechny mají charakteristiky (vlastnosti) a možnosti (metody) předdefinovaného MovieClip objektu. Časové osy mají specifický vztah k sobě navzájem v závislosti na umístění v seznamu zobrazení. Časové osy, které jsou umístěny uvnitř dalších Časových os a jsou ovlivňovány změnami, ke kterým dojde na Časové ose ve které se nacházejí. Například, jestliže je **`portland`** dítě **`oregonu`** a vy změníte vlastnost **`_xscale oregonu`**, **`portland`** se změní také.

Časové osy si mohou navzájem posílat zprávy. Například akce na posledním snímku jednoho movie klipu může spustit přehrávání dalšího movie klipu.

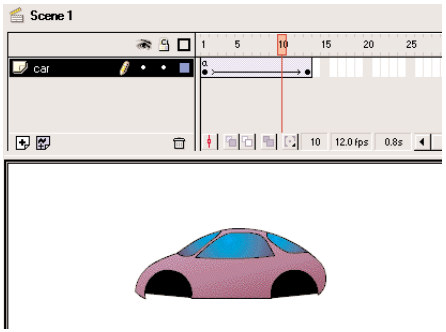
O hierarchickém vztahu Časových os

Jestliže umístíte movie klip instanci na Časovou osu jiného movie klipu, tak vlastně jeden symbol movie klipu obsahuje instanci jiného movie klipu - první movie klip je **dítě** a druhý movie klip je **rodič**. Hlavní Časová osa Flash animace je **rodič** (parent) všech movie klipů na této úrovni.

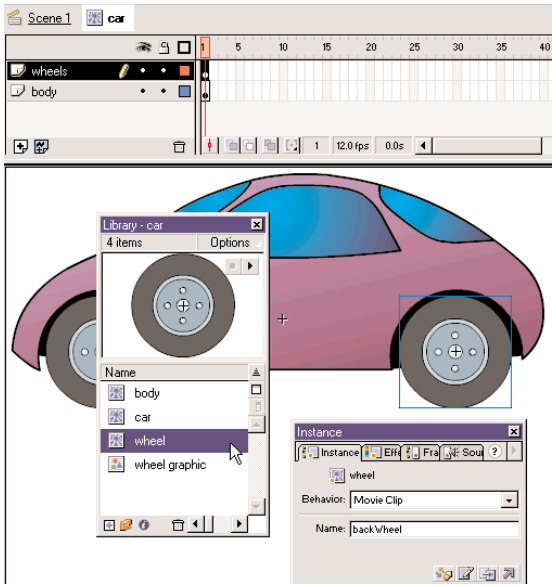
Vztahy rodič-dítě movie klipů jsou hierarchické. Pro pochopení této hierarchie si představte hierarchii počítače: hard disk má základní adresář (nebo složku) a podadresáře. Základní adresář je shodný s hlavní Časovou osou v animaci: je to rodič všeho ostatního. Podadresáře jsou analogické movie klipům. Podadresáře můžete použít, pro organizaci příbuzných obsahů.

Jakákoliv změna v rodičovském movie klipu je provedena také na jeho dítěti.

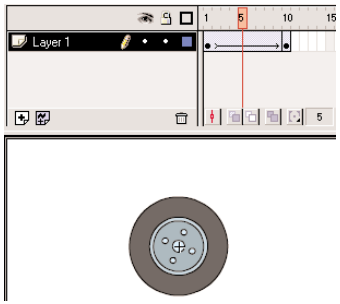
Například byste mohli vytvořit animaci, ve které jede auto přes Plochu. Mohli byste použít movie klip symbol, který reprezentuje auto a nastavit jeho pohyb přes Plochu.



Na auto je pohlíženo z boku, kde má dvě viditelná kola. Při pohybu auta chcete, aby kola rotovala. Takže uděláte movie klip pro kolo auta a vytvoříte dvě instance tohoto movie klipu, pojmenované **frontWheel** a **backWheel**. Potom umístíte kola na Časovou osu movie klipu **car** - ne na hlavní Časovou osu. Jako děti **car** jsou **frontWheel** a **backWheel** ovlivněny změnami, které nastanou u auta. To znamená, že se při pohybu auta přes plochu budou pohybovat s autem.



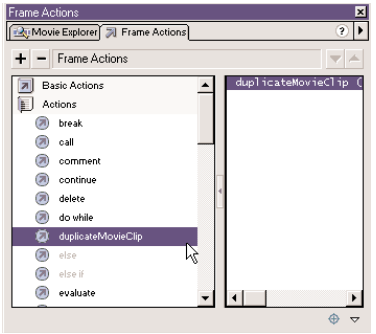
Rotující kola vytvoříte nastavením motion tween na rotaci symbolu kola, a zároveň také na rotaci obou instancí. Dokonce poté, co změníte **frontWheel** a **backWheel**, interpolace jejich rodičovského movie klipu **car** budou nadále ovlivněny; kola se budou točit, ale budou se také pohybovat s rodičovským klipem **car** přes Plochu.

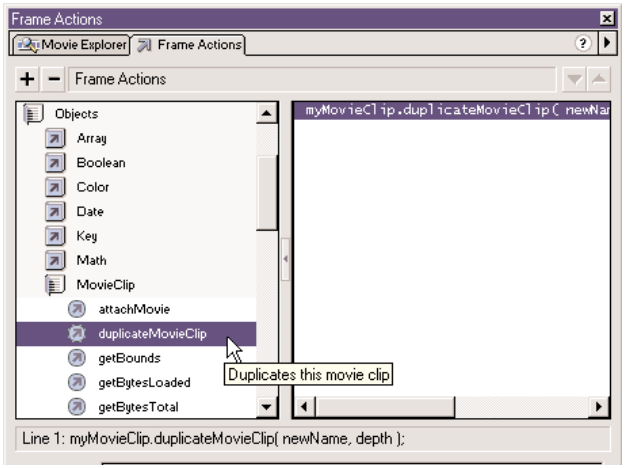


Posílání zpráv mezi Časovými osami

Zprávy můžete posílat z jedné Časové osy do druhé. Jedna Časová osa obsahuje akci, nazvanou **controller** a druhá, která obdrží akci nazvanou **target**. Akce můžete přidělit v Časové ose ke snímku, tlačítku, nebo movie klipu.

Pro cílování Časových os můžete použít akce z kategorie Action, nebo můžete použít metody MovieClip objektu z kategorie Objects v panelu Akcí. Například při přehrávání animace můžete použít akci **duplicateMovieClip** pro určitý klip-cíl a vytvořit tak kopie instancí movie klipu.





Při provádění několika akcí ve stejném cíli použijte akci **with**. Podobně jako u JavaScript příkazu **with**, akce ActionScriptu **with** je obal, který umožňuje adresovat cílovou Časovou osu pouze jednou a potom vykonat série akcí v tomto klipu; nemusíte adresovat cílovou Časovou osu v každé akci.

K provedení několika akcích ve stejném cíli také můžete použít akci **`tellTarget`**.

Pro komunikaci mezi Časovými osami musíte udělat následující:

- Vložte jméno instance cílového movie klipu.

Pro pojmenování movie klip instance použijte Panel Instance (Window > Panels > Instance). Časové osy natažené do úrovní používají svá čísla úrovní jako jména instancí, například **`_level6`**.

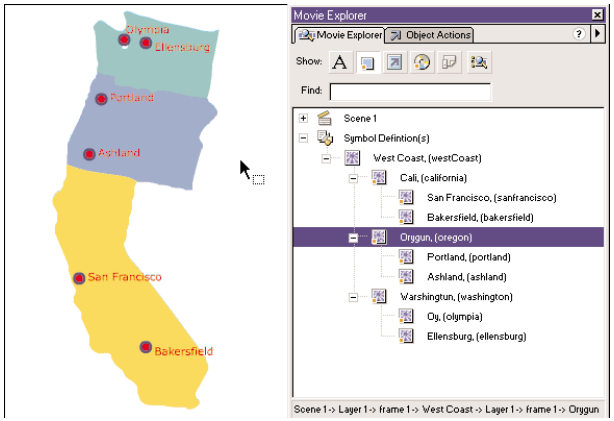
Vložte cílovou cestu do jména instance v Panelu Akce.

Cílovou cestu můžete vložit ručně nebo lze použít dialog box Insert Target Path. Viz „Specifikace cílových cest“.

Poznámka: Aby mohla být cílována Časová osa movie klipu, musí být během přehrávání na Scéně.

O absolutních a relativních cílových cestách

Cílová cesta je adresa Časové osy, která je cílována. Seznam zobrazení Časových os ve Flashi je podobný hierarchii souborů a složek na Web serveru.



Stejně jako na Web serveru, každá Časová osa ve Flashi může být adresována dvěma způsoby: absolutní cestou nebo relativní cestou. Absolutní cesta instance je vždy stejná, bez ohledu na to, jakou Časovou osu akce volá; například absolutní cesta k instanci **`california`** je vždy **`_level0westCoast.california`**. Relativní cesta je různá, protože bývá volána z různých umístění; například relativní cesta do california ze sanfrancisco je **`_parent`**, ale z **`portland`** je **`_parent._ parent.california`**.

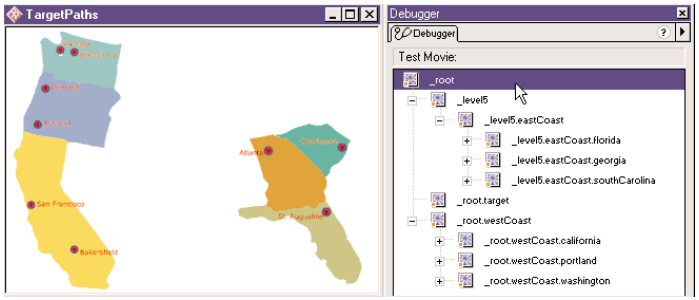
Poznámka: Více informací o Movie Explorer viz **Using Flash.**

Absolutní cesta začíná jménem úrovně, do které je animace natažena a pokračuje přes seznam zobrazení až dosáhne cílové instance.

(Pro potřeby této knihy chápejte animace jako .swf soubory)

První animace, která má být otevřena ve Flash Přehrávači je natažena na úroveň 0. Ke každé dodatečně natažené animaci musíte připojit číslo úrovně. Cílové jméno pro úroveň je **`_levelX`**, kde **`X`** je číslo úrovně, do které je animace natažena. Například první animace otevřená ve Flash Přehrávači je nazvána **`_level0`**, animace natažená do úrovně 3 je nazvána **`_level3`**.

V následujícím příkladě byly dvě animace nataženy do přehrávače, **`TargetPaths.swf`** do úrovně 0 a **`EastCoast.swf`** do úrovně 5. Úrovně jsou indikovány v Debuggeru, s úrovní 0 indikovanou jako **`_root`**.



Instance má vždy nějakou absolutní cestu, ať je volána z akce v instanci na stejné úrovni nebo z akce na jiné úrovni. Například instance **`bakersfield`** na úrovni 0, má v dot syntaxi vždy následující absolutní cestu v dot syntaxi:
`_level0.california.bakersfield`

Ve slash syntaxi jsou tečky v absolutní cestě nahrazeny lomítky:
`_level0/california/bakersfield`

Pro komunikaci mezi animacemi na různých úrovních musíte použít jméno úrovně v cílové cestě. Například instance **`portland`** by adresovala instanci **`atlanta`** následovně:
`_level5.georgia.atlanta`

V dot syntaxi můžete použít alias **_root** pro odkaz na hlavní Časovou osu aktuální úrovně. Pro hlavní Časovou osu, nebo **_ level0**, alias **_root** reprezentuje **_level0**, když je cílován klipem také na **_level0**. Pro animaci nataženou do **_level5**, je **_root** roven **_level5**, pokud je cílována movie klipem také na úrovni 1. Například akce volaná z instance **southcarolina** by mohla použít následující absolutní cestu pro cílování instance **florida**:

```
_root.eastCoast.florida
```

Ve slash syntaxi můžete použít / pro odkaz na hlavní Časovou osu aktuální úrovně:

```
/eastCoast/florida
```

V dot syntaxi, v Absolutním i v Relativním Režimu, můžete použít stejná pravidla cílování pro identifikaci proměnné na Časové ose, nebo vlastnosti objektu. Například následující příkaz nastavujte jméno proměnné ve formě instance k hodnotě „Gilbert“:

```
_root.form.name = „Gilbert“;
```

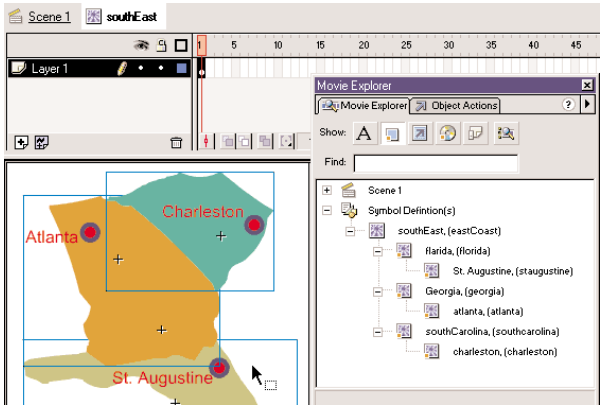
Ve slash syntaxi, v Absolutním i v Relativním Režimu, můžete identifikovat proměnnou na Časové ose tak, že jméno proměnné předchází dvojtečka ():

```
/form:name = „Gilbert“;
```

Relativní cesta je závislá na vztahu mezi kontrolorem Časové osy a cílovou Časovou osou. Relativní cestu můžete využít pro znovupoužití akcí, protože stejná akce může být cílována do různých Časových os v závislosti na umístění akce. Relativní cesty mohou adresovat cíle pouze uvnitř své vlastní úrovně Flash Přehrávače; nemohou adresovat animace natažené do jiných úrovní. Například nemůžete použít relativní cestu v akci na **_level0**, která cíluje Časovou osu na **_level5**.

V dot syntaxi, v relativní cílové cestě, pro odkaz na aktuální Časovou osu můžete použít klíčové slovo **this**. V relativní cílové cestě, pro indikování rodičovské Časové osy aktuální Časové osy, můžete použít alias **_parent**. Alias **_parent** může být použit opakovaně pro skok o jednu úroveň výš v movie klip hierarchii, uvnitř stejné úrovně Flash Přehrávače. Například **_ parent._parent** kontroluje movie klip o dvě úrovně výš v hierarchii.

V následujícím příkladu je každé město (**charleston, atlanta** a **staugustine**) dítětem instance stát a každý stát (**southcarolina, georgia** a **florida**) je dítě instance **eastCoast**.



Akce na Časové ose instance **charleston**, by mohla použít následující cílovou cestu pro cílování instance **southcarolina**:

```
_parent
```

Pro cílování instance **eastCoast** z akce v **charleston**, byste mohli použít následující relativní cestu:

```
_parent._parent
```

Ve slash syntaxi můžete použít dvě tečky (**..**), abyste se dostali o úroveň výš v hierarchii. Pro cílování **eastCoast** z akce v **charleston** byste mohli použít následující cestu:

```
../..
```

V dot syntaxi, pro cílování instance **atlanta** z akce na Časové ose **charlestonu**, byste mohli použít následující relativní cestu:

```
_parent._parent.georgia.atlanta
```

Relativní cesty jsou užitečné pro znovupoužití scénářů. Například byste mohli skript připojit k movie klipu, který jej zvětšuje o 150%:

```
onClipEvent (load){
    _parent._xscale = 150;
    _parent._yscale = 150;
}
```

Potom byste tento skript mohli znovu použít jeho umístěním do Časové osy jakéhokoliv jiného movie klipu.

Více informací o adresování a dot syntaxi viz „Psaní Scénářů s ActionScript“.

Více informací o Dot syntaxi a Slash syntaxi viz „Použití ActionScript syntaxe“.

Specifikování cílových cest

Ke kontrole movie klipu nebo natažené animace musíte k určení cíle použít cílovou cestu. Aby mohl být movie klip cílován, musí mít jméno instance. Cíl můžete určit několika různými způsoby:

- Vložte cílovou cestu pomocí použití tlačítka Insert Target Path a dialog boxu v panelu Akcí.
- Vložte cílovou cestu movie klipu ve skriptu ručně.
- Vytvořte výraz použitím odkazu na movie klip, nebo použitím předdefinovaných funkcí **`targetPath`** a **`eval`**.

Vložení cílové cesty pomocí použití dialog boxu Insert Target Path:

1. Zvolte movie klip, snímek nebo instanci tlačítka, ke kterému chcete připojit akci.

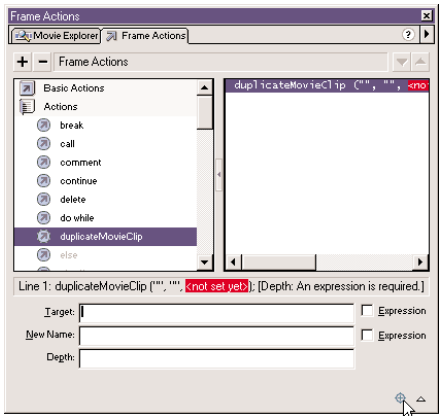
   To bude kontrolor Časové osy.

2. Zvolte Window > Actions pro zobrazení panelu Akcí.

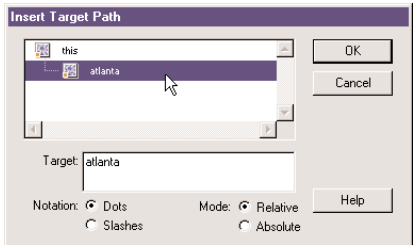
3. V seznamu Toolbox list zvolte akci, nebo metodu z kategorie MovieClip uvnitř složky Objects.

4. Klikněte na pole Target, nebo umístění ve skriptu pro vložení cílové cesty.

5. Klikněte na tlačítko Insert Target Path v pravém dolním rohu panelu Akcí, pro zobrazení dialog boxu Insert Target Path.



6. V dialog boxu Insert target Path zvolte syntaxi: Dots (nastaveno) nebo Slashes.



7. Zvolte Absolutní nebo Relativní pro režim cílové cesty.

   Viz „O absolutní a relativních cílových cestách“.

8. Určete váš cíl jedním z následujících způsobů:
   - Zvolte movie klip v seznamu zobrazení **`Insert Target Path.`**
   - Vložte cíl ručně do pole Target pomocí použití absolutní nebo relativní cesty a dot syntaxe.

9. Klikněte OK.

Vložení cílové cesty ručně:
Proveďte kroky 1-4 nahoře a vložte absolutní nebo relativní cestu do panelu Akce.

Použití výrazu jako cílové cesty:
1. Proveďte kroky 1-4 nahoře.

2. Udělejte jedno z následujících:

   - Ručně vložte odkaz na cílovou cestu. Odkaz pro určení cílové cesty je ohodnocen. Odkaz můžete použít jako parametr pro akci **with**. V následujícím příkladě je proměnná **index** ohodnocena a vynásobena 2. Výsledná hodnota je použita jako jméno movie klipu uvnitř instance **Block**, které je řečeno, aby hrála:

     ```
     with (Board.Block [index*2]){
           play ();
     }
     ```

   - V kategorii Functions seznamu Toolbox list zvolte funkci **targetPath**. Tato funkce konvertuje odkaz na movie klip do řetězce, který může být použitý akcemi jako **tellTarget**.

     V následujícím příkladě funkce **targetPath** konvertuje odkaz **Board.Blok [index*2+1]** na řetězec:

     ```
     tellTarget (targetPath (Board.Blok[index*2+1])){
           play ();
     }
     ```

Předchozí příklad je ekvivalentní následující Slash syntaxi:

```
tellTarget ("Board/Block:"+index*2+1)){
   play();
}
```

   - V kategorii Functions, v seznamu Toolbox list zvolte funkci **eval**.

     Funkce **eval** konvertuje řetězec na odkaz na movie klip, který může být použit jako cílová cesta akcemi jako **with**.

     Následující scénář ohodnocuje proměnnou **i**, přidává ji k řetězci **„cat"** a připojuje výslednou hodnotu k proměnné **x**. Proměnná **x** je nyní odkaz na movie klip instanci a může volat metody MovieClip objektu:

     ```
     x = eval („cat"+i);
     x.play();
     ```


Funkci **eval** můžete také použít pro přímé volání metod:

```
eval („cat"+i).play();
```

Používání akcí a metod pro kontrolu Časových os

Pro cílování, nebo pro provedení úkolů v movie klipu nebo natažené úrovni, můžete použít určité akce a metody MovieClip objektu. Například akce **setProperty** nastavuje vlastnost (jako je **_width**) Časové osy na nějakou hodnotu (např. 100). Některé metody MovieClip objektu duplikují funkci všech akcí, používaných k cílování Časové osy. Existují také dodatkové metody, jako je **hitTest** a **swapDepth**. Když použijete akci nebo metodu a chcete ji zároveň volat, musí být cílová Časová osa natažená do Flash Přehrávače.

Následující akce mohou cílovat movie klipy: **loadMovie, unloadMovie, setProperty, startDrag, duplicateMovieClip** a **removeMovieClip**. Při použití těchto akcí musíte vložit cílovou cestu, do parametru akce Target zadat příjemce akce. Některé z těchto akcí mohou cílovat movie klipy nebo úrovně a jiné mohou cílovat pouze movie klipy.

Následující metody objektu MovieClip mohou kontrolovat movie klipy nebo natažené úrovně a nemají ekvivalentní akce: **attachMovie, getBounds, getBytesLoaded, getBytesTotal, globalToLocal, lokalToGlobal, hitTest** a **swapDepth.**

Pokud akce a metoda nabízí podobné funkce, můžete zvolit pro kontrolu movie klipů kterýkoliv z nich. Volba je závislá pouze na vašich schopnostech a znalostech skriptování.

Více informací o metodách MovieClip objektu a informace o každé akci viz Kapitola 7, „ActionScript Slovník".

Metody versus akce

Metody voláváte použitím cílové cesty ke jménu instance, následovaném tečkou a potom jménem metody a argumentu, jako v následujících příkazech:

```
myMovieClip.play();
parentClip.childClip.gotoAndPlay(3);
```

V prvním příkazu metoda **play** způsobí, že instance **myMovieClip** bude hrát. Ve druhém příkazu metoda **gotoAndPlay** posílá hrací hlavu na **childClip** (který je dítětem instance **parentClip**) na snímek 3 a hraje.

Akce, které kontrolují Časovou osu mají parametr Target, určující cílovou cestu. Například v následujícím scénáři akce **startDrag** cíluje instanci **customCursor** a činí jej tažitelný:

```
on(press){
   startDrag("customCursor");
}
```

Když použijete metodu, voláte ji až na konci cílové cesty. Například následující příklad prování stejnou funkci **startDrag**:

```
customCursor.startDrag();
```

Příkazy napsané s použitím metod objektu MovieClip jsou stručnější, protože nepožadují akci **tellTarget**. Od použití akce **tellTarget** je odrazováno, protože není kompatibilní s ECMA-262 standardem. Například ke spuštění přehrávání movie klipu **myMovieClip** pomocí metod objektu MovieClip, byste použili následující kód:

```
myMovieClip.play();
```

Následující kód dává stejný výsledek, ale s použitím akce **tellTarget**:

```
tellTarget ("myMovieClip"){
    play();
}
```

Použití několika metod nebo akcí pro cílování Časové osy

Akci **with** stačí použít pro cílování movie klipu jen jednou, potom můžete vykonávat sérii akcí na tomto klipu. Akce **with** funguje ve všech objektech ActionScriptu (například Array, Color a Sound, ne pouze movie klipech. Akce **tellTarget** je podobná akci **with**, nicméně není preferována, protože nefunguje se všemi objekty ActionScriptu a nevyhovuje ECMA-262.

Akce **with** bere objekt jako parametr. Objekt, který určíte je připojen na konec aktuální cílové cesty. Všechny akce vložené do akce **with**, jsou vykonány uvnitř nové cílové cesty nebo **pole působnosti**. Například v následujícím skriptu na hlavní Časové ose je akce **with** přiřazena objektu **donut.hole**, aby změnila vlastnosti **hole**:

```
with (donut.hole){
    _alpha = 20;
    _xscale = 150;
    _yscale = 150;
}
```

Je to, jako kdyby příkazy uvnitř akce **with** byly volány z Časové osy instance **hole**.

V následujícím příkladu si všimněte hospodárnosti použití akce **with** a metod MovieClip objektu pro stanovení několika instrukcí:

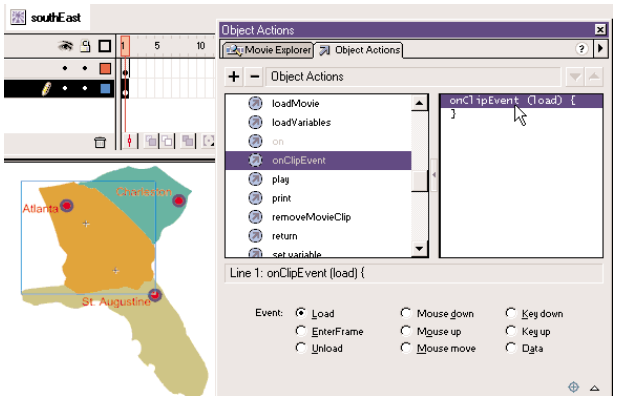
```
with (myMovieClip){
    _x -= 10;
    _y += 10;
    gotoAndPlay(3);
}
```

Více informací o akci **tellTarget** viz **Using Flash**.

Připojení akce nebo metody

Akce a metody mohou být připojeny k tlačítku, nebo snímku na Časové ose, nebo k instanci movie klipu.

Pro připojení akce nebo metody k instanci movie klipu musíte použít ovladač **onClipEvent**. Všechny akce připojené k instanci jsou vloženy do ovladače **onClipEvent** a jsou vykonány ihned po jejich spuštění. Akce **onClipEvent** je spouštěna buď událostmi Časové osy (jako je natažení animace) nebo událostmi uživatele (jako je kliknutí myši nebo stisknutí klávesy). Například **onClipEvent** (**mouseMove**) spouští akci pokaždé, když uživatel pohne s myší.



Natažení a stažení externích animací

Akci nebo metodu **loadMovie** můžete použít k natažení a přehrání externích animací, nebo k vypnutí animaci bez natahování další HTML stránky. Tuto akci nebo metodu lze také použít pro poslání proměnných do CGI skriptu, který generuje SWF soubor jako jeho CGI výstup. Při natahování animace, můžete určit úroveň nebo cílový movie klip, do kterého bude animace natažena.

Akce a metoda **unloadMovie** odstraňuje dříve nataženou animaci pomocí **loadMovie**. Explicitní stažení animace pomocí **unloadMovie** zajišťuje hladký přechod mezi animacemi a může také odlehčit paměti požadované Flash Přehrávačem. Akci **loadMovie** použijte pro vytvoření některého z následujících příkladů:

- Přehrávání sekvence reklamních banerů (SWF souborů), umístěním akce **loadMovie** na konec každého SWF souboru, pro natažení další animace
- Vyviňte rozvětvené rozhraní, které uživateli umožní vybrat si mezi několika různými SWF soubory.
- Na úrovni 0 vytvořte navigační rozhraní s navigačními ovladači, které nahrají ostatní úrovně.

Změna pozice a vzhledu movie klipu

Pro změnu vlastností movie klipu při jejich přehrávání můžete použít akci **setProperty** nebo napsat příkaz, který určujte hodnotu proměnné. Jestliže natahujete animaci do cíle, natažená animace zdědí vlastnosti cílovaného movie klipu. Pokud už je animace natažena, můžete tyto vlastnosti změnit.

Některé vlastnosti, nazvané **pouze pro čtení**, mají hodnoty, které můžete číst, ale ne nastavit. Z toho vyplývá, že psát příkazy pro nastavení vlastností lze pouze pro ty, které nejsou určeny pouze pro čtení. Následující příkaz nastavuje vlastnost **_alpha** instanci movie klipu **wheel**, která je dítětem instance **car**:

```
car.wheel._alpha = 50;
```

Navíc, můžete psát příkazy, které získávají hodnotu vlastnosti movie klipu. Například, následující příkaz dostává hodnotu vlastnosti **_xmouse** na hlavní Časové ose a nastavujte na tuto hodnotu vlastnost **_x** instance **customCursor**:

```
onClipEvent(enterFrame){
    customCursor._x = _root._xmouse;
}
```

Získání vlastnosti movie klipu také zajistí použití funkce **getProperty**.

Vlastnosti **_x**, **_y**, **_rotation**, **_xscale**, **_yscale**, **_height**, **_width**, **_alpha** a **_visible** jsou ovlivňovány transformacemi na rodičovském movie klipu a tím transformují také movie klip jakéhokoliv dítěte. Vlastnosti **_focusrect, _highquality, _quality** a **_sounbuftime** jsou globální; patří pouze k úrovni 0 Časové osy. Všechny ostatní vlastnosti patří ke každému movie klipu nebo natažené úrovni. V tabulce dole je seznam všech movie klip vlastností:

Vlastnost	
_alpha	_target
_currentframe	_totalframes
_droptarget	_url
_focusrect	_visible
_framesloaded	_width
_height	_x
_highquality	_xmouse
_name	_xscale
_quality	_y
_rotation	_ymouse
_soundbuftime	_yscale

Tažení movie klipů

Akce nebo metoda **startDrag**, učiní při přehrávání animace movie klip tažitelný. Tažení movie klipu má využití ve hrách, ve funkcích táhni a pusť, speciálním rozhraní, v rolujících sloupcích a diapozitivech. Movie klip zůstává tažitelný, dokud není explicitně zastaven pomocí **stopDrag**, nebo dokud není jiný klip cílován pomocí **startDrag**. V jednom čase může být tažitelný pouze jeden klip.

Při tvorbě komplikovanějších chování táhni a pusť můžete ohodnocovat tažený movie klip vlastností **_droptarget**. Tuto vlastnost můžete vyzkoušet při tažení klipu na specifický klip (např.movie klip „trash can“) a po uvolnění klipu spustit další akci. Viz „Použití výrazů „if“ a „Použití operátorů pro manipulaci s hodnotami ve výrazech“.

Duplikování a odstraňování movie klipů

Instanci movie klipu můžete vytvořit nebo odstranit při přehrávání animace, pomocí použití **duplicateMovieClip**, nebo **removeMovieClip**. Akce a metoda **duplicateMovieClip** dynamicky vytváří novou instanci movie klipu, připojuje jí nové jméno instance a dává jí hloubku. Duplikovaný movie klip vždy začíná na snímku 1, i když původní movie klip byl v okamžiku duplikování na jiném snímku, a je stále na vrcholu všech předdefinovaných movie klipů umístěných na Časové ose. Do duplikovaného movie klipu nejsou kopírovány proměnné.

Pro odstranění movie klipu, který jste vytvořili pomocí **duplicateMovieClip**, použijte **removeMovieClip**. Když je odstraněn rodičovský movie klip, duplikované movie klipy jsou odstraněny také.

Připojení movie klipů

Kopii movie klipu můžete získat z knihovny a přehrávat ji jako část animace lze pomocí metody **attachMovie**. Tato metoda během hraní animace natáhne jiný movie klip do vašeho movie klipu a přehrává jej.

Při použití metody **attachMovie** musí být připojovaný movie klip pojmenován jedinečným jménem v dialog boxu Symbol Linkage Properties.

Pojmenování movie klipu pro sdílení:

1. V Knihovně animace zvolte movie klip, který chcete připojit.
2. V okně Knihovna zvolte Linkage z menu Options.
3. Pro Linkage zvolte Export This Symbol.
4. Do dialog boxu Symbol Linkage Properties, vložte jméno movie klipu do okna pro Identifier. Jméno se musí lišit od jmen symbolů v knihovně.
5. Klikněte OK.

Připojení movie klipu k dalšímu movie klipu:

1. V panelu Akce určete cíl, ke kterému chcete připojit movie klip.

2. V seznamu Toolbox list zvolte objekt MovieClip a potom vyberte metodu **`attachMovie`**.

3. Nastavte následující argumenty:
   - Pro **`idName`** specifikujte jméno, které jste vložili do dialog boxu Symbol Linkage Properties, do pole Identifier.
   - Pro **`newName`** vložte jméno instance pro připojovaný klip tak, abyste ho mohli cílovat.
   - Pro **`depth`** vložte úroveň, na kterou má být k movie klipu připojen duplikovaný klip. Připojené movie klipy jsou vždy na vrcholu původního movie klipu.

Například:

```
myMovieClip.attachMovie(„calif", „california",10);
```

Vytváření „smart“ klipů

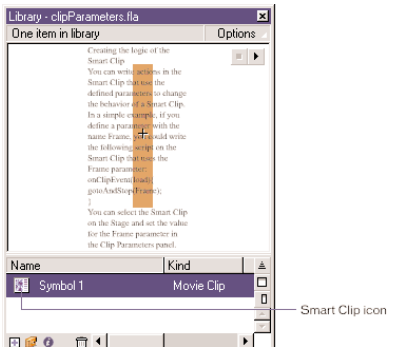
„Smart“ klip je movie klip s definovanými parametry, které mohou být měněny. Tyto parametry jsou vkládány do akcí měnících jeho chování.

Při vytváření smart klipu určíte jeho parametry v Knihovně, v symbolu movie klipu. Do smart klipu můžete napsat příkazy ActionScriptu, které operující s jeho parametry, podobně jako používáte argumenty v definici funkcí. Můžete zvolit instanci smart klipu na Scéně a změnit hodnoty jejích parametrů v panelu Klip Parametry. Během přehrávání, jsou hodnoty nastavené v panelu posílány do smart klipu, před jakýmikoliv akcemi vykonávanými v animaci.

Smart klipy jsou užitečné pro předávání komplikovaných prvků od programátora k návrháři. Programátor může do smart klipu napsat akce s proměnnými, které kontrolují klip a animaci. Návrhář potom může změnit hodnoty těchto proměnných v panelu Klip Parametry aniž by otevíral panel Akcí.
Smart klipy můžete použít pro vytváření prvků rozhraní - jako jsou rádiová tlačítka, pop-up menu, uživatelské tipy, průzkumy, hry a variantní fáze trvalé entity. Jakýkoliv klip animace, který chcete později jinak použít bez změny skriptů bude dobrým smart klipem.
Dále můžete vytvářet přizpůsobené rozhraní ve Flashi pro panel Klip Parametry, abyste usnadnili práci designérům, kteří klip upravují.

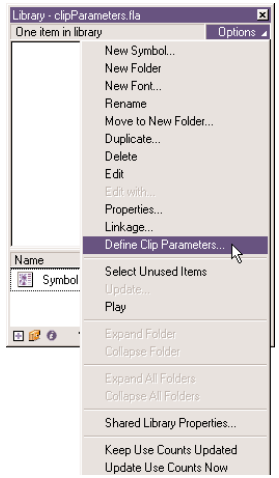
Definování parametrů klipu

Parametry klipu jsou části dat, které jsou vkládány do movie klipu, při jeho natahování do animace. Parametry klipu můžete definovat už při jeho tvorbě. Během přehrávání animace pomocí parametrů můžete měnit vzhled a chování smart klipu. Movie klip s definovanými parametry indikuje v Knihovně Speciální ikona.



Definování parametrů pro movie klip:

1. Zvolte symbol movie klip v knihovně a udělejte jednu z následujících věcí pro zobrazení dialog boxu Clip Parameters:
   - Klikněte vpravo (Windows) nebo na Control (Macintosh) a zvolte Define Clip Parameters z kontext menu.
   - Zvolte Define Clip Parameters z menu Options v vpravo nahoře v okně Knihovna.



2. Použijte kontroly v dialog boxu Clip Parameters následovně:
   - Klikněte na tlačítko Add (+) pro přidání nové dvojice name/value (jméno/hodnota), nebo dodatečných parametrů pro vybraný pár jméno/hodnota.
   - Klikněte na tlačítko Minus (-) pro zrušení dvojice jméno/hodnota.
   - Použijte tlačítka šipek pro změnu pořadí parametrů v seznamu.
   - Zvolte pole tak, že na něj dvakrát kliknete, a potom vložte hodnotu.
3. Do Name vložte unikátní identifikátor parametru.
4. Do Type zvolte z pop-up menu druh dat, které bude parametr obsahovat:
   - Zvolte Default pro použití řetězcové nebo číselné hodnoty.
   - Zvolte Arrow pro dynamický seznam položek, které mohou růst nebo se zmenšovat.
   - Zvolte Object pro deklarování několika vztahujících se prvků se jmény a hodnotami, jako je objekt Point s prvky **x** a **y**.
   - Zvolte List pro omezení výběru na několik voleb jako je **true** nebo **false** nebo **Red**, **Green** nebo **Blue**.
5. Pro Value zvolte z pop-up menu nastavenou hodnotu, kterou bude parametr obsahovat.
6. Pokud chcete použít upravené prostředí pro panel Klip Parametry, proveďte jedno z následujících:
   - Zadejte relativní cestu do SWF soubor s upraveným rozhraním v poli Link to Custom UI (Odkázat na upravené UI).
   - Klikněte na složku Link to Custom UI a zobrazte SWF s upraveným rozhraním.

     Viz „Vytváření speciálního rozhraní“.
7. Pro Description vložte poznámky, které se objeví v panelu Klip Parametry a které popisují, co každý parametr dělá.
   - Do Popisu můžete zahrnout jakoukoliv informaci podle které se může někdo další v klipu orientovat. Například vysvětlení metod, které jste definovali atd.

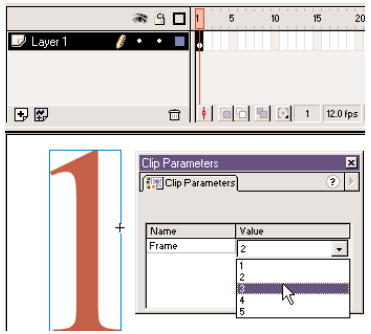
8. Zvolte Lock in Instance, abyste zabránili uživatelům v přejmenování parametrů v panelu Klip Parametry.
   Je doporučováno, abyste nechali jména parametrů zamknutá.
9. Klikněte OK.

Nastavení parametrů klipu

Do smart klipu můžete psát akce, které používají definované parametry, abyste změnili chování chytrého klipu. Jednoduše, pokud definujete parametr klipu pojmenovaný Frame, můžete do chytrého klipu, který používá parametr **Frame**, zapsat následující skript:

```
onClipEvent(load){
    gotoAndStop(Frame);
}
```

Potom můžete vybrat Smart Klip na Scéně a nastavit hodnotu parametru **Frame** v panelu Klip Parametry pro změnu aktuálně přehrávaného snímku.



Nastavení parametrů smart klipu:

1. Zvolte instanci smart klipu na Scéně.
   - Smart klipy jsou movie klipy, takže v autorském režimu je zobrazen pouze první snímek.
2. Zvolte Window > Panels > Clip Parameters, pro zobrazení panelu Klip Parametry.

3. V panelu Klip Parametry udělejte jedno z následujících:
   - Dvakrát klikněte na pole Value pro jeho vybrání a vložte hodnotu pro všechny parametry.

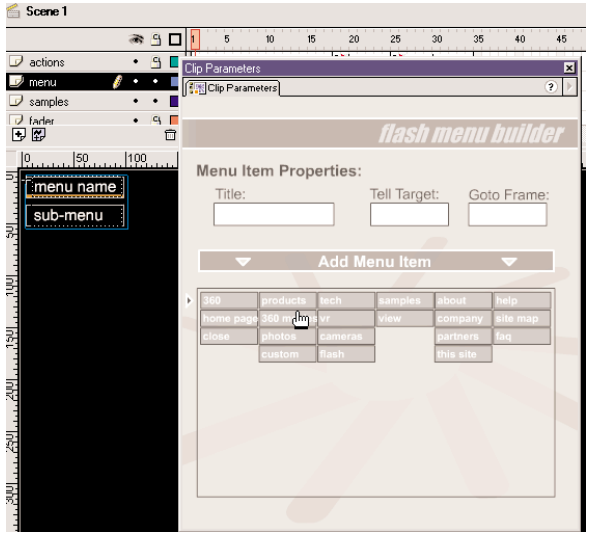
   Jestliže byl parametr definován jako List, objeví se pop-up menu.

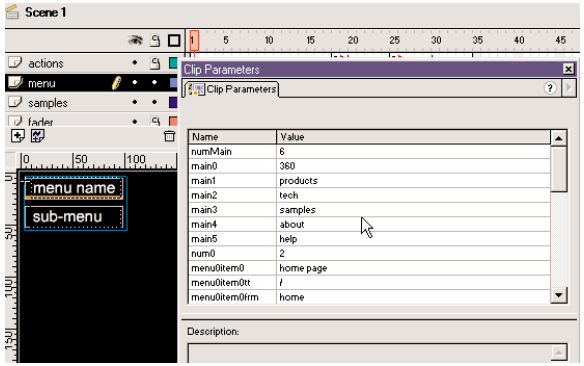
   - Jestliže bylo definováno speciální rozhraní, použijte poskytnuté prvky rozhraní.

4. Zvolte Control > Test Movie, abyste viděli změnu chování smart klipu.

Vytváření speciálního rozhraní

Custom interface (Speciální rozhraní) je Flash animace, která umožňuje vložit hodnoty do smart klipu. Speciální rozhraní nahrazuje rozhraní panelu Klip Parametry.





Jakékoliv hodnoty, které zadáte s použitím speciálního rozhraní, jsou poslány z panelu Klip Parametry klipu smart klipu přes zprostředkující, neboli výměnný, klip animace ve speciálním rozhraní. Výměnný klip animace musí mít název instance **`xch`**. Pokud je v dialogovém okně Define Clip Parameters označeno speciální rozhraní, pošle instance smart klipu definované parametry do klipu animace **`xch`** a jakékoliv nově zadané hodnoty ve speciálním prostředí jsou zkopírovány do **`xch`** a poslány zpět smart klipu.

Klip **`xch`** musíte umístit na hlavní Časovou osu animace rozhraní a také **`xch`** musí být stále natažen. Movie klip xch by měl obsahovat pouze hodnoty, které mají být vkládány do Smart Klipu. Neměl by obsahovat jakoukoliv grafiku, jiné movie klipy nebo příkazy ActionScriptu; **`xch`** je pouze schránka, přes kterou jsou posílány hodnoty. Neměli byste ale předávat Arrays nebo Objects.

Vytvoření speciálního rozhraní pro Smart klip:

1. Zvolte File > New pro vytvoření nového Flash movie.

2. Zvolte Insert > New Symbol pro vytvoření výměnného movie klipu.

3. Vytvořte novou vrstvu nazvanou „Exchange Clip“.

4. S vybranou vrstvou „Exchange Clip“ táhněte výměnný movie klip z okna Knihovna na Scénu ve snímku 1.

5. Zvolte výměnný movie klip na Scéně, zvolte Window > Panels > Instance a vložte jméno **`xch`**.

6. Vytvořte prvky rozhraní, se kterými bude autor interaktivovat s nastavením klip parametrů. Například pop-up menu, radio tlačítka nebo položky menu táhni a pusť.

7. Použijte akci **`set variable`** pro kopírování proměnné a hodnot objektu do instance **`xch`**.

   Například, jestliže je tlačítko použito jako prvek rozhraní, tlačítko by mohlo mít akci, která nastavuje hodnotu proměnné vertical a propouští ji do xch:

```
on (release){
    _root.xch.vertical = true;
}
```

8. Exportujte movie jako SWF soubor.

Abyste mohli použít SWF se speciálním rozhraním se Smart Klipem, potřebujete je spojit v dialogovém okně Define Clip Parameters v knihovně, které obsahuje Smart Klip. Doporučuje se uložit si SWF soubor do stejného adresáře jako FLA obsahujícího Smart Klip. Pokud Smart Klip použijete v jiném souboru nebo jej pošlete jinému vývojáři, musí Smart Klip a SWF se speciálním rozhraním zůstat ve stejných relativních umístěních.

KAPITOLA 5

Integrace Flash s Web Aplikacemi

Flash animace mohou posílat informace do a natahovat informace ze vzdálených souborů. Pro posílání a natahování proměnných používáte akci **`loadVariables`**, nebo **`getURL`**. Pro natažení animace (SWF souboru) ze vzdáleného umístění používáte akci **`loadMovie`**. Pro posílání a natahování XML dat používáte objekt XML nebo XML Socket. XML data můžete strukturovat pomocí metod předdefinovaných XML objektů.

Můžete také vytvářet formy Flashe skládající se z přizpůsobených prvků rozhraní, jako jsou textová pole a pop-up menu, abyste sbírali data, která budou odeslána aplikaci na straně serveru.
Abyste Flash rozšířili tak, že by odesílal vzkazy a také je přijímal z prostředí animace hostitele - například Přehrávače Flashe nebo funkce JavaScript v internetovém prohlížeči – můžete použít **`fscommand`** a metody Přehrávače Flashe.

Posílání a natahování proměnných do a ze vzdáleného souboru

Flash animace je vlastně okno pro získávání a zobrazování informací, podobně jako HTML stránka. Flash animace, na rozdíl od HTML stránek, mohou zůstat nataženy v prohlížeči a nepřetržitě aktualizovat informace. Pro posílání informací do a obdržení informace ze serverových skriptů, textových souborů a XML souborů, můžete použít akce a metody Flash objektů. Serverové skripty mohou požadovat specifickou informaci z databáze a předávat ji zpět a vpřed mezi databází a Flash animací. Serverové skripty mohou být napsány v různých jazycích: nejběžnější jsou Perl, ASP (Microsoft Active Server Pages) a PHP.

Ukládání informací v databázi a jejich získávání umožňuje vytvářet dynamické animace. Například můžete vytvořit tabuli zpráv, personální profily pro uživatele nebo nákupní kartu, která si pamatuje co uživatel koupil, takže může určit preference uživatele.

Každá akce a metoda, která přenáší informace z a do animace používá pro přenos informací protokol. Každá také požaduje určitý způsob formátování informace.

Následující akce používají HTTP nebo HTTPS protokol pro posílání informace v URL zakódovaném formátu: **`getURL, loadVariables, loadMovie`**.

Následující metody používají HTTP nebo HTTPS protokol pro poslání informace jako XML: **`XML.send, XML.load, XML.sendAndLoad`**.

Následující metody vytvářejí a používají TCP/IP socket spojení pro poslání informace jako XML: **`XMLSocket.connect, XMLSocket.send`**.

O bezpečnosti

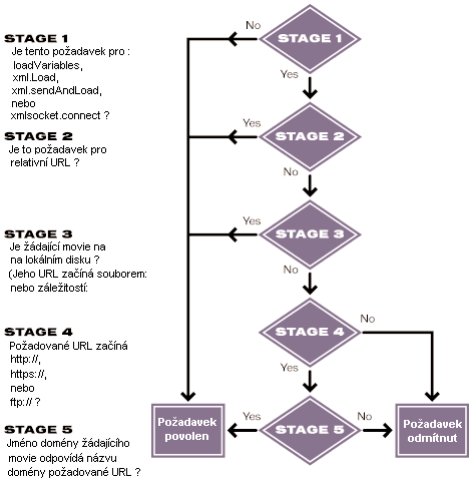
Když přehráváte Flash animaci ve webovém prohlížeči, můžete do animace nahrát data pouze ze souboru, který je na serveru ve stejné subdomémě. To zabrání tomu, aby byly animace ve Flashi schopny stahovat informace ze serverů jiných lidí.
Abyste určili subdoménu URL skládajícího se z jedné nebo dvou složek, použijte celou doménu:

Doména	Subdoména
`http://macromedia`	`macromedia`
`http://macromedia.com`	`macromedia.com`

Pro určení subdomény URL sestávající z více než dvou komponent odstraňte poslední úroveň:

Doména	Subdoména
`http://x.y.macromedia.com`	`y.macromedia.com`
`http://x.y.macromedia.com`	`macromedia.com`

Následující graf ukazuje, jak Flash Přehrávač určuje, zda připustit nebo nepřipustit HTTP požadavek:



Když používáte XML Socket objekt pro vytvoření socket spojení se serverem, musíte použít port číslo 1024 nebo vyšší. (Porty s nižším číslem jsou běžně používány pro Telnet, FTP, World Wide Web nebo Finger.)

Flash se spoléhá na bezpečnostní rysy standardního browseru, HTTP a HTTPS. V podstatě Flash nabízí stejnou bezpečnost, která je dosažitelná u standardního HTML. Měli byste dodržovat stejná pravidla, která používáte při zabezpečování HTML stránek. Například pro zajištění bezpečného hesla ve Flashi potřebujete nastavit autentikaci hesla s žádostí na web server.

Pro vytvoření hesla použijte textové pole, které bude heslo od uživatele požadovat. Předložte ho serveru v akci **`loadVariables`** nebo v metodě **`XML.sendAndLoad`**, pomocí použití HTTPS URL s metodou POST. Web server může potom verifikovat, zda je heslo platné. Tímto způsobem nebude heslo nikdy dostupné v SWF souboru.

Kontrola natažených dat

Každá akce a metoda, která natahuje data do animace (kromě **`XMLSocket.send`**) je **asynchronní**; výsledky akce jsou obdrženy v neurčeném čase.

Dříve než budete moci natažená data v animaci použít, musíte provést kontolu, zda byla data natažena. Například natáhnout proměnné a manipulovat s hodnotami těchto proměnných můžete v jednom skriptu. V následujícím skriptu nemůžete použít proměnnou **`lastFrameVisited`**, dokud si nejste jisti, že byla natažena ze souboru **`myData.txt`**:

```
loadVariables („myData.txt“,0);
gotoAndPlay (lastFrameVisited);
```

Každá akce a metoda má určitou techniku, kterou lze použít pro kontrolu natažených dat. Jestliže použijete akce **`loadVariables`** nebo **`loadMovie`**, můžete natáhnout informaci do cíle movie klipu a pro vykonání skriptu použít událost **`data`**, akce **`onClipEvent`**. Jestliže použijete akci **`loadVariables`** pro natažení dat, akce **`onClipEvent(data)`** se vykoná až když je natažena poslední proměnná. Jestliže použijete akci **`loadMovie`** pro natažení dat, akce **`onClipEvent`**(data) se vykoná i když je natažena pouze část animace.

Následující akce tlačítka, natáhne proměnné ze souboru **`myData.txt`** do movie klipu **`loadTargetMC`**:

```
on(release){
    loadVariables(„myData.txt“, _root.loadTargetMC);
}
```

Akce připojená k instanci **`loadTargetMC`** používá proměnnou **`lastFrameVisited`**, která je natažena ze souboru **`myData.txt`**. Následující akce se vykoná poté, co jsou všechny proměnné, včetně **`lastFrameVisited`**, nataženy:

```
onClipEvent(data){
    gotoAndPlay(lastFrameVisited);
}
```

Jestliže použijete metody **XML.load** a **XMLSocket.connect**, můžete nadefinovat ovladač, který postoupí data ihned po jejich obdržení. Ovladač je vlastnost XML nebo XMLSocket objektu, ke kterému připojíte nadefinovanou funkci. Ovladače jsou volány automaticky při obdržení informace. Pro XML objekt použijte **XML.onLoad**. Pro **XMLSocket** objekt použijte **XMLSocket.onConnect**.

Více informací viz „Použití objektu XML“ a „Použití objektu XMLSocket“.

Použití loadVariables, getURL a loadMovie

Všechny akce **loadVariables, getURL** a **loadMovie** komunikují se serverovými skripty pomocí HTTP protokolu. Každá akce posílá všechny proměnné z Časové osy, ke které je tato akce připojena; akce reagují následovně:

- **getURL** dává jakoukoliv informaci do okna browseru, nebo do Flash Přehrávače.
- **loadVariables** natahuje proměnné do určené Časové osy ve Flash Přehrávači.
- **loadMovie** natahuje animaci do určené úrovně ve Flash Přehrávači.

Když používáte akce **loadVariables**, **getURL** nebo **loadMovie**, můžete zadat několik argumentů:

- **URL** je soubor obsahující vzdálené proměnné.
- **Location** je úroveň, nebo cíl v animaci která proměnné obdrží.

Více informací o úrovních a cílech viz „O několika Časových osách“.

Poznámka: Akce **getURL** argument this nepotřebuje.

- **Variables** nastavuje HTTP způsob, buď GET nebo POST, kterým budou proměnné poslány.

Například, jestliže chcete sledovat nejvyšší skóre ve hře, měli byste ukládat skóre na serveru a použít akci **loadVariables** pro jejich natažení do animace při každém spuštění hryy. Akce by mohla vypadat takto:

```
loadVariables („http://www.mojeStranka.cz/skripty/vysoke_skore.php“,
_root.skoreKlip,GET);
```


Toto natáhne proměnné z PHP skriptu nazvaného **vysoke_skore.php** do instance movie klipu **skoreKlip** pomocí HTTP způsobu **GET**.

Jakékoliv proměnné natažené pomocí akce **loadVariables** musí být ve standardní MIME formát aplikaci/x-www-urlformencoded (standardní formát používaný CGI skripty). Aby Flash mohl soubor číst, musí být soubor který určujete v URL argumentu akce **loadVariables** zapsán dvojicí proměnná a hodnota, v tomto formátu.

Soubor může určit jakýkoliv počet proměnných; dvojice proměnná a hodnota musí být odděleny znaménkem (**&**) a slova uvnitř hodnoty musí být oddělena znaménkem plus (**+**). Například tato fráze definuje několik proměnných:

```
vysokeSkore1=54000&hracJmeno1=rockin+dobry&vysokeSkore2=53455&hrac
Jmeno2=bonehelmet&vysokeSkore3=42885&hracJmeno3=soda+pop
```

Více informací o **loadVariables, getURL** a **loadMovie** viz Kapitola 7, „ActionScript Slovník.“

O XML

XML (**Extensible Markup Language**) se stává standardem pro výměnu strukturovaných dat v Internetových aplikacích. Data ve Flashi můžete integrovat se servery, které používají XML technologii pro vytváření sofistikovaných aplikací, jako je chat systém nebo makléřský systém atd.

V XML, tak jako v HTML, používáte tagy pro **markup** (označení), nebo určení těla textu. V HTML používáte předdefinované tagy pro indikaci, jak by měl text vypadat v prohlížeči (například tag **<b>** indikuje, že text by měl být tučný). V XML definujete tagy, které identifikují typ části dat (například **<heslo>VelmiTajné</heslo>**). XML odděluje strukturu informace od způsobu, jakým je zobrazena. Toto umožňuje, aby byl stejný XML dokument použitý a znovu použitý v různých prostředích.

Každý XML tag je nazván **node** nebo prvek. Každý node má typ (1-XML prvek nebo 3-text node) a také prvky mohou mít atributy. Node „uhnízděný“ v node je nazván **child** (dítě) nebo **childNode** (dětskýNode). Takováto hierarchická stromová struktura nodů je nazvaná XML DOM (Document Object Model) - podobně jako DOM JavaScriptu, který je strukturou prvků ve Webovém prohlížeči.

V následujícím příkladu, je **<PORTFOLIO>** rodičovský node; nemá žádné atributy a obsahuje dětskýNode **<HOLDING>**, který má atributy SYMBOL, QTY, CENA a HODNOTA:

```
<PORTFOLIO>
    <HOLDING SYMBOL=„BOHATÝ“
        QTY=„75“
        CENA=„245.50“
        HODNOTA=„18412.50“/>
</PORTFOLIO>
```

Použití objektu XML

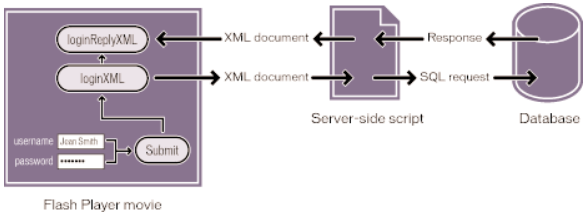
Metody ActionScript objektu XML můžete použít (například **appendChild, removeNode** a **insertBefore**), pro strukturování XML dat ve Flashi, pro poslání stažených XML dat na server, manipulaci s nimi a jejich interpretaci.

Následující metody objektu XML můžete použít pro poslání a natažení XML dat na server přes HTTP způsob POST:

- **load** natáhne XML z URL a umístí ho do ActionScript XML objektu.
- **send** předá XML objekt do URL. Jakákoliv vrácená informace je poslána do dalšího okna browseru.
- **sendAndLoad** pošle XML objekt do URL. Jakákoliv vrácená informace je umístěna do ActionScript XML objektu.

Například byste mohli vytvořit makléřský systém pro obchodování s cennými papíry, který ukládá všechny informace (jména uživatelů, hesla, ID zasedání, portfolio holdingy a informace o transakcích) do databáze.

Serverový skript, který předává informaci mezi Flash a databází, čte a píše data v XML formátu. ActionScript můžete použít, pro konverzi informace získané ve Flash animaci (například username a password) na XML objekt a potom data poslat do serverového skriptu jako XML dokument. Také můžete použít ActionScript pro natažení XML dokumentu, který server vrací do XML objektu, pro použití v animaci.



Validace heslem pro makléřský systém požaduje dva skripty: funkci definovanou na snímku 1 a skript, který vytváří a posílá XML objekty připojené k tlačítku Submit ve formuláři.

Když uživatelé vloží svou informaci do textových polí ve Flash animaci s proměnnými **username** a **password**, musí být proměnné konvertovány na XML dříve než budou předány na server. První část skriptu natáhne proměnné do nově vytvořeného XML objektu nazvaného **loginXML**. Když uživatel stiskne tlačítko Submit, je objekt **loginXML** konvertován na řetězec XML a poslán na server.

Následující skript je připojen k tlačítku Submit. Pro pochopení skriptu čtěte komentáře indikované znaky //:

```
on (release){
    // A.Konstrukce XML dokumentu a prvkem LOGIN
    loginXML = new XML();
    loginElement = loginXML.createElement("LOGIN");
    loginElement.attributes.username = username;
    loginElement.attributes.password = password;
    loginXML.appendChild(loginElement);

    // B.Konstrukce XML objektu pro obdržení odpovědi serveru
    loginReplyXML = new XML();
    loginReplyXML.onLoad = onLoginReply;

    // C.Poslání prvku LOGIN do serveru, umístění odpovědí do loginReplyXML
    loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",
    loginReplyXML);
}
```

První část skriptu generuje následující XML při uživatelově stisku tlačítka SUBMIT:

```
<LOGIN USERNAME="JeanSmith"PASSWORD="VelmiTajné"/>
```

Server obdrží XML, generuje XML odpověď a posílá ji zpět do Flash animace. Jestliže je heslo akceptováno, server odpoví následujícím způsobem:

```
<LOGINREPLY STATUS="OK"SESSION="rnr6f7vkj2oe14m7jkkycilb"/>
```

Tento XML zahrnuje atribut SESSION, který obsahuje unikátní, náhodně generované session ID, které bude použito ve všech komunikacích mezi klientem a serverem po celý zbytek spojení. Jestliže je heslo odmítnuto, server odpoví následující zprávou:

```
<LOGINREPLY STATUS="FAILED"/>
```

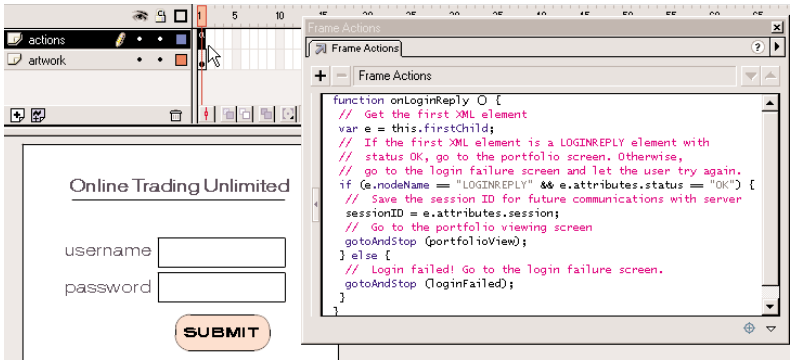
XML node **LOGINREPLY** se musí natáhnout do prázdného XML objektu ve Flash animaci.

Následující příkaz vytváří XML objekt **loginreplyXML** pro obdržení XML node:

```
//B. Konstrukce XML objektu pro obdržení odpovědi serveru
loginReplyXML = new XML();
loginReplyXML.onLoad = onLoginReply;
```

Druhý příkaz připojuje funkci **onLoginReply** k ovladači **loginreplyXML.onLoad**.

XML prvek **LOGINREPLY** přichází asynchronně, podobně jako data z akce **loadVariables**, a natahuje se do objektu **loginReplyXML**. Když data přijdou, je volána metoda **onLoad** objektu **loginReplyXML**. Musíte definovat funkci **onloginReply** a připojit ji k ovladači **loginReplyXML.onLoad** tak, aby mohla předávat prvek **LOGINREPLY**. Funkce **onloginReply** je připojena ke snímku, který obsahuje tlačítko **SUBMIT**.



Funkce **onloginReply** je definována v prvním snímku animace. Pro pochopení skriptu čtěte komentáře, které jsou označeny znaky //:

```
function onLoginReply(){
    // Získat první XML prvek
    var e = this.firstChild;
    // Jestliže je první XML prvek prvek LOGINODPOVĚĎ se
    // stavem OK, jdi na obrazovku portfolio. Jinak,
   // jdi na obrazovku login neúspěšné a nech to uživatele zkusit znovu.
    if (e.nodeName == "LOGINREPLY" && e.attributes.status == "OK"){
// Uložení ID session pro budoucí komunikaci se serverem
    sessionID = e.attributes.session;
// Jdi na obrazovku prohlížení portfolia
        gotoAndStop("portfolioView");
    }else {
        // Login neúspěšné! Jdi na obrazovku neúspěšné login.
        gotoAndStop("loginFailed");
    }
}
```


První řádek této funkce, **var e=this.firstChild**, používá klíčové slovo **this** pro odkaz na XML objekt **loginReplyXML**, který byl právě natažen s XML ze serveru. Můžete použít this, protože **onloginReply** bylo voláno jako **loginReplyXML.onLoad**, takže i když se zdá, že **onloginReply** vypadá jako jednoduchá funkce, ve skutečnosti se chová jako metoda **loginReplyXML**.

Pro poslání username a password jako XML do serveru a natažení XML odpovědi zpět do Flash animace, můžete použít metodu **sendAndLoad**:

```
//C. Poslat prvek LOGIN do serveru, umístit odpověď do loginReplyXML
loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",
loginReplyXML);
```

Více informací o XML metodách viz Kapitola 7, „ActionScript Slovník.“

Poznámka: Tento návrh je pouze příklad a neklade žádné požadavky na úroveň bezpečnosti. Jestliže implementujete bezpečnostní systém chráněný heslem, ujistěte se, že jste dobře pochopili bezpečnost sítě.

Použití objektu XMLSocket

ActionScript nabízí předdefinovaný XMLSocket objekt, který umožňuje otevřít nepřetržité spojení se serverem. Socket spojení umožňuje serveru předávat informace ke klientovi, jakmile je informace dostupná. Bez nepřetržitého spojení musí server čekat na požadavek HTTP. Toto otevřené spojení odstraňuje latentní problémy a je běžně používáno pro aplikace v reálném čase, jako jsou chaty. Data jsou posílána přes socket spojení jako jeden řetězec a měla by být v XML formátu. Pro strukturování dat můžete použít XML objekt.

Při tvorbě socket spojení musíte vytvořit serverovou aplikaci, která bude čekat na požadavek socket spojení a posílat odpověď do Flash animace. Tento typ serverové aplikace může být napsán v programovacím jazyce Java.

Metody objektu XMLSocket **connect** a **send**, můžete použít pro transfer XML do a ze serveru přes socket spojení. Metoda **connect** zřizuje socket spojení s portem Web serveru. Metoda **send** posílá XML objekt na server specifikovaný v socket spojení.

Jestliže voláváte metodu **connect**, Flash Přehrávač otevře TCP/IP spojení se serverem a udržuje spojení otevřené, dokud se nestane následující:

- Je volána metoda **close** objektu XMLSocket.
- Neexistují už žádné odkazy na objekt XMLSocket.
- Je ukončena aplikace Flash Přehrávače.
- Je přerušeno spojení (například nespojení nebo odpojení modemu).

Následující příklad vytváří XML socket spojení a posílá data z XML objektu myXML. Pro pochopení skriptu čtěte komentáře indikované znaky//:

```
//vytvořit nový XMLSocket objekt
sock = new XMLSocket();
//volat jeho metodu connect ke zřízení spojení s portem 1024 serveru na URL
sock.connect("http://www.myserver.com",1024);
//definovat funkci pro připojení sock objektu, který řídí odpověď serveru.
//Jestliže je spojení úspěšné, poslat myXML objekt.
//Jestliže úspěšné není, poskytnout chybovou zprávu v textovém poli.
function onSockConnect(success){
    if (success){
        sock.send(myXML);
    }else {
        msg = "Je zde chybné spojení se" + jménoServeru;
    }
}
//připojit funkci onSockConnect k vlastnosti onConnect
sock.onConnect = onSockConnect;
```

Více informací viz Kapitola 7, „ActionScript Slovník.“

Vytváření formulářů

Formuláře poskytují pokročilý typ interaktivity - kombinace tlačítek, animací a textových polí, které umožňují předávání informací do jiné aplikace na lokálním nebo vzdáleném serveru. Všechny běžné prvky formulářů (jako jsou radio tlačítka, drop-down seznamy a check boxy) mohou být vytvořeny jako movie klipy nebo tlačítka, která zapadnou do struktury vašich stránek, dle vašeho grafického cítění, nebo podle designu vaší Web stránky. Nejběžnějším prvkem formuláře je vstupní (input) textové pole.

Běžné typy formulářů, které používají prvky rozhraní, zahrnují chat rozhraní, objednávkové formuláře a prohledávací rozhraní. Formulář může například sbírat informace o adresách a posílat je do jiné aplikace, která kompiluje informace do e-mail zprávy nebo databázového souboru. Dokonce i jednoduché textové pole je považováno za formulář a může být použito pro sbírání vstupů uživatele a zobrazení výsledků.

Formuláře požadují dvě hlavní komponenty: prvky Flash rozhraní, které vytvářejí formulář a buď serverovou aplikaci nebo klientský skript pro předávání informace vložené uživatelem. Následující kroky ukazují všeobecný postup pro vytváření formuláře ve Flash.

Vytvoření formuláře:

1. Umístěte prvky rozhraní do animace, do určené vstvy.

   Můžete použít prvky rozhraní z běžné knihovny Buttons-Advanded, nebo vytvořit svoje vlastní.

2. V panelu Volby Textu nastavte textová pole na Input a každému připojte unikátní jméno proměnné.

   Více informací o vytváření editovatelných textových polí viz **Using Flash**.

3. Připojte akci, která buď posílá, natahuje nebo posílá a natahuje data.

Vytváření prohledávacího formuláře

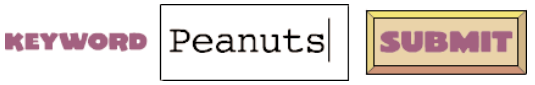
Příklad jednoduchého formuláře je prohledávací pole s tlačítkem Submit. Jako úvod do vytváření formulářů poskytuje následující příklad instrukce pro vytváření prohledávacího rozhraní pomocí použití akce **`getURL`**. Vložením požadované informace mohou uživatelé poslat klíčové slovo do prohledávacího stroje na vzdáleném Web serveru.

Vytvoření jednoduchého prohledávacího formuláře:

1. Vytvořte tlačítko pro předání vložených dat.

2. Vytvořte popisek, prázdné textové pole a instanci tlačítka na Scéně.

   Vaše obrazovka by měla vypadat takto:



3. Zvolte textové pole a zvolte Window > Panels > Text Option.

4. V panelu Volby Textu nastavte následující volby:
   - Zvolte Input Text z pop-up menu.
   - Zatrhněte Border/Bg.
   - Určete jméno proměnné.

Poznámka: Jednotlivé prohledávací stroje mohou požadovat specifické jméno proměnné. Detaily jsou uvedené ve Web site prohledávací stroje.

5. Na Scéně zvolte tlačítko a zvolte Window > Action.

Objeví se panel Objekt Akce.

Poznámka: Kontrolka vedle Akce v menu Window indikuje, že je panel otevřený.

6. Přetáhněte akci **getURL** z boxu nástrojů do okna Skriptu.

7. V panelu Parametry nastavte následující volby:

   - Pro URL vložte URL prohledávacího stroje.
   - Pro Window zvolte **_blank**. Toto otevře nové okno, které zobrazí výsledky prohledávání.

Pro Variable zvolte Send using GET.

8. Pro testování formuláře zvolte File > Publish Preview > HTML.

Použití proměnných ve formulářích

Proměnné ve formulářích můžete použít pro uložení vstupu uživatele. Pro posílání proměnných použijte editovatelná textová pole nebo v prvcích rozhraní připojte akce k tlačítkům. Například každá položka v pop-up menu je tlačítko s akcí, která nastavuje proměnnou pro indikování vybrané položky. Můžete připojit jméno proměnné do vstupu textového pole. Textové pole se pak chová jako okno, které zobrazuje hodnotu této proměnné.

Když posíláte informaci do a ze serverového skriptu, musí proměnné ve Flash animaci souhlasit s proměnnými ve skriptu. Například, jestliže skript očekává proměnnou nazvanou **password**, textové pole, do kterého budou uživatelé vkládat heslo, by mělo mít proměnnou jménem **password**.

Některé skripty požadují schované (hidden) proměnné, což jsou proměnné, které uživatel nikdy neuvidí. Pro vytvoření schovaných proměnných ve Flashi můžete nastavit proměnnou na snímku v movie klipu, který obsahuje prvky jiného formuláře. Schované proměnné jsou poslány do serverového skriptu spolu s ostatními proměnnými nastavenými na Časové ose obsahující akci, která předkládá formulář.

Verifikace vložených dat

Ve formuláři, který propouští proměnné do aplikace na Webovém serveru budete chtít verifikovat, že uživatelé vkládají správnou informaci. Například nechcete, aby uživatelé vkládali text do pole telefonní číslo. Použijte sérii akcí **set variable** ve spojení s **for a if**, pro ohodnocení vložených dat.

Následující ukázková akce kontroluje, zda je vložené číslo a že číslo je ve formátu *###-###-####*. Jestliže jsou data platná, je zobrazena zpráva „Toto je platné telefonní číslo!“. Jestliže data nejsou platná, je zobrazena zpráva „Toto telefonní číslo je neplatné!“.

Pro použití tohoto skriptu v animaci vytvořte dvě textová pole na Scéně a pro každé zvolte Input v panelu Volby Textu. Připojte proměnnou **phoneNumber** k jednomu textovému poli a ke druhému připojte proměnnou **message**. Připojte následující akci k tlačítku na Scéně vedle textových polí:

```
on (release){
    valid = validPhoneNumber(phoneNumber);
    if (valid){
        message = "Toto je platné telefonní číslo!";
    }else {
        message ="Toto telefonní číslo je neplatné!";
    }
    function isdigit(ch){
        return ch.length == 1 && ch >= '0' && ch <= '9';
    }
    function validPhoneNumber(phoneNumber){
        if (phoneNumber.length != 12){
            return false;
        }
        for (var index = 0; index < 12; index++){
            var ch = phoneNumber.charAt(index);
            if (index == 3 ||index == 7){
                if (ch != "-"){
                return false;
                }
            }else if (!isdigit(ch)){
                return false;
            }
        }
        return true;
    }
}
```

Pro poslání dat vytvořte tlačítko, které má akci podobnou následující:
(Nahraďte argumenty **getURL** argumenty, které odpovídají vaší animaci.)

```
on (release){
    if (valid){
        getURL("http://www.webserver.com", "_self", "GET");
    }
}
```

Více informací o těchto ActionScript příkazech viz **set**, **for** a **if** v Kapitole 7, „ActionScript Slovník“.

Posílání zpráv do a z Flash Přehrávače

Pro posílání zpráv z animace do jejího hostitelského prostředí (například Webového browseru, Director movie nebo samostatného Flash Přehrávače) můžete použít akci **fscommand**. To umožní rozšířit animaci o použití schopnosti hostitele. Například byste mohli poslat akci **fscommand** do funkce JavaScript na HTML stránce, která otevírá okno browseru s určitými vlastnostmi.

Pro ovládání animace ve Flash Přehrávači ze skriptovacích jazyků prohlížeče jako JavaScript, VBScript a Microsoft Jscript můžete použít metody Flash Přehrávače -- funkce, které posílají zprávy z hostitelského prostředí do Flash animace. Například byste mohli mít řádek na HTML stránce, který posílá vaši Flash animaci na určitý snímek.

Použití fscommand

Použijte akci **fscommand** pro poslání zprávy do jakéhokoliv programu, který hostí Flash Přehrávač. Akce **fscommand** má dva parametry: **command** (příkaz) a **argumets** (argumenty). Při posílání zprávy do samostatné verze Flash Přehrávače musíte použít předdefinované příkazy a argumenty. Například následující akce nastavuje samostatný přehrávač pro rozšíření animace na velikost celé obrazovky, když je uvolněno tlačítko:

```
on(release){
    fscommand("fullscreen", "true");
}
```


Následující tabulka ukazuje hodnoty, které můžete zadat do parametrů příkaz a argumenty akce **fscommand**, pro kontrolu animace hrající v samostatném přehrávači (včetně projektorů):

Command	Arguments	Účel
quit	Žádné	Zavírá projektor.
fullscreen	**true / false**	Specifikace **true** nastavuje **Flash** Přehrávač na režim celé obrazovky. Specifikace false nechává přehrávač v normálním režimu zobrazení.
allowscale	**true / false**	Specifikace **false** nastavuje přehrávač tak, že animace je stále přehrávána ve své původní velikosti a není nikdy přizpůsobována. Specifikace **true** přizpůsobí animaci na 100% přehrávače.
showmenu	**true / false**	Specifikace **true** umožní plné nastavení položek kontext menu. Specifikace **false** zastíní všechny položky kontext menu kromě O Flash Přehrávači.
exec	Cesta do aplikace	Vykoná aplikaci z uvnitř projektoru.

Při použití **fscommand** k posílání zprávy do skriptovacího jazyka jako JavaScript ve Web browseru, můžete předávat jakékoliv dva argumenty v parametrech **command** a **arguments**. Tyto argumenty mohou být řetězce nebo výrazy a budou použity v JavaScript funkci, která „chytá“ nebo řídí akci **fscommand**.

Akce **fscommand** volá na HTML stránce JavaScript funkci **moviename_DoFSCommand**, která vkládá Flash animaci, kde **moviename** je jméno Flash Přehrávače, které je určeno atributem **NAME** tagu EMBED, nebo atributem **ID** tagu **OBJECT**. Jestliže je Flash Přehrávač určen jménem **myMovie**, volaná JavaScript funkce je **myMovie_DoFSCommand**.

Použití akce fscommand, pro otevření boxu zpráv z Flash animace na HTML stránce přes JavaScript:

1. Na HTML stránku, která vkládá Flash animaci připojte následující JavaScript kód:

```
function theMovie_DoFSCommand(command, args){
    if (command == "messagebox"){
        alert(args);
    }
}
```

Jestliže publikujete animaci pomocí Flashe se šablonou FSCommand v HTML Nastavení Publikace, je tento kód vložen automaticky. Atributy movie **NAME** a ID budou jména souborů. Například pro soubor **myMovie.fla** by byly atributy nastaveny na **myMovie**. Více informací od publikování viz **Using Flash**.

2. Ve Flash animaci přidejte akci **fscommand** k tlačítku:
   fscommand("messagebox","Toto je box zpráv volávaných z Flash.")

Také můžete použít výrazy pro akci **fscommand** a argumenty:
fscommand("messagebox","Ahoj,"& name &", vítej na naší Web site!")

3. Zvolte File > Publiblish Preview > HTML, pro otestování movie.

Akce **fscommand** může do Macromedia Directoru poslat zprávy, které jsou interpretovány pomocí Lingo jako řetězce, události nebo proveditelný Lingo kód. Jestliže je zpráva řetězec nebo událost, musíte napsat Lingo kód pro její obdržení z akce **fscommand** a vykonat akci v Directoru. Více informací viz Director Support Center na http://www.macromedia.com/support/director.

Ve Visual Basic, Visual C++ a dalších programech, které mohou hostit ActiveX kontroly, posílá **fscommand** VB událost se dvěma řetězci, které mohou být ovládány v prostředí programovacího jazyka. Pro více informací použijte klíčová slova **Flash method** pro hledání Flash Support Center na http://www.macromedia.com/support/flash.

O metodách Flash Přehrávače

Metody Flash Přehrávače můžete použít pro ovládání animace ve Flash Přehrávači ze skriptovacích jazyků Web prohlížeče, jako jsou JavaScript a VBScript. Metody lze také použít pro posílání volání do Flash Přehrávače z jiných skriptovacích prostředí než je ActionScript. Každá metoda má jméno a většina metod také argumenty. Argument specifikuje hodnotu, na které metoda operuje. Kalkulace provedená některými metodami dává hodnotu, která může být použita skriptovacím prostředím.

Existují dvě různé technologie umožňující komunikaci mezi prohlížečem a Flash Přehrávačem: LiveConnect (Netscape Navigator 3.0 nebo později na Windows 95/98/2000/NT nebo Power Macintosh) a ActiveX (Microsoft Internet Explorer 3.0 a později na Windows 95/98/2000/NT). Ačkoliv techniky skriptování jsou podobné pro všechny prohlížeče a jazyky, jsou zde dodatečné vlastnosti a události dostupné pro použití s kontrolami ActiveX.

Pro získání více informací, včetně kompletního seznamu skriptovacích způsobů Flash Přehrávače použijte klíčová slova **Flash method** pro vyhledání Flash Support Center na http://www.macromedia.com/support/flash.

KAPITOLA 6

Odstraňování chyb v ActionScriptu

Úroveň sofistikovanosti (úroveň složitosti) některých akcí, zvláště v kombinaci s dalšími, může vytvářet vysokou obtížnost skriptů ve Flash animacích. Stejně jako u jiných programovacích jazyků, můžete napsat nesprávný skript, který způsobuje chyby v animaci. Použití správných autorských technik usnadňuje odstraňování závad v animaci, pokud se něco chová neočekávaně.

Flash má několik nástrojů pro testování animace v Režimu Test-Movie nebo ve Web browseru. Debugger (Odstraňovač závad) ukazuje hierarchický seznam zobrazení movie klipů aktuálně natažených do Flash Přehrávače. Umožňuje také zobrazit a upravovat hodnoty proměnné, během přehrávání animace. V Režimu Test-Movie, okno Output zobrazuje chybové zprávy, seznamy proměnných a objektů. Také můžete ve skriptech použít akci trace, pro poslání programovacích poznámek a hodnot výrazů do okna Output.

Autorské směrnice a směrnice pro odstraňování závad

Jestliže používáte správné autorské praktiky při psaní skriptů, budou vaše animace mít málo chyb (programovacích chyb). Snažte se dodržovat následující zásady, které vám pomohou zabránit vzniku problémů, a v případě jejich výskytu je i rychle nalézt.

Použití správných autorských praktik

Je dobré během práce ukládat několik verzí animace. Zvolte File > Save As pro uložení verze s jiným jménem každé půl hodiny. Odpověď na otázku kde začal problém naleznete sledováním historie ukládání jednotlivých verzí tím, že naleznete poslední soubor, ve kterém ještě problém nebyl. Při použití tohoto postupu máte stále funkční verzi, dokonce i když je jeden soubor špatný.

Další důležitou autorskou praktikou je testovat včas, testovat často a testovat na všech cílových platformách, abyste nalezli problémy hned po jejich výskytu. Zvolte Control > Test Movie, pro přehrání animace v režimu test-movie, kdykoliv uděláte větší změnu nebo před uložením verze. V režimu test-movie běží animace ve verzi samostatného přehrávače.

Pokud je vaše animace určená pro Web, je důležité ji testovat také v prohlížeči. V určitých situacích (například jestliže vyvíjíte pro intranet) možná budete znát typ prohlížeče a platformu uživatele. Jestliže vyvíjíte pro Web, testujte animaci ve všech prohlížečích na všech potencionálních platformách.

Je dobré používat tyto autorské praktiky:

- Používejte akci **trace** pro posílání komentářů do okna Output. Viz „Použití stopa“.

- Používejte akci **comment** pro zahrnutí instruktážních poznámek, které se objeví pouze v panelu Akce. Viz „Komentáře“.

- Používejte odpovídající konvence pojmenovávání pro identifikaci prvků ve skriptu. Například je dobré vyhnout se mezerám ve jménech. Začínejte jména proměnných a funkcí malým písmenem a používejte velké písmeno pro každé nové slovo **(mojePromennaJmeno, mojeFunkceJmeno)**. Začínejte jména konstrukční funkce velkým písmenem **(MojeKonstrukcniFunkce)**. Nejdůležitější je zvolit styl, který vám dává smysl a důsledně ho používat.

- Používejte jména proměnných, která mají význam odrážející druh informace, kterou proměnná obsahuje. Například proměnná obsahující informaci a naposledy stisknutém tlačítku by mohla být pojmenována **posledniStisknuteTlacitko**. Jméno jako **foo** by těžko připomenul čeho se proměnná týká.

- Používejte editovatelná textová pole ve vodicích (guide) vrstvách pro sledování hodnot proměnných jako alternativu k použití Debuggeru.

- Používejte Movie Explorer v Režimu Edit-Movie pro prohlížení seznamu zobrazení a prohlížení všech akcí v animaci. Viz Flash Help.

- Používejte akci **for…in**, pro smyčkování napříč vlastnostmi movie klipů, včetně dětských movie klipů. Akci **for…in** s akcí **trace** můžete použít pro poslání seznamu vlastností do okna Output. Viz „Opakování akce“.

Použití kontrolních seznamů odstraňování závad

Stejně jako u každého skriptovacího prostředí i zde tvůrci skriptů běžně dělají určité chyby. Následující seznam je dobrý pro začátek práce při odstraňování problémů v animaci:

- Ujistěte se, že jste v režimu test-movie.

  V autorském prostředí budou fungovat pouze jednoduchá tlačítka a akce snímku (například **gotoAndPlay** a **stop**). Zvolte Control > Enable Simple Frame Actions nebo zvolte Control > Enable Simple Buttons, pro umožnění těchto akcí.

- Ujistěte se, že nemáte akce snímku na několika vrstvách, které jsou mezi sebou v rozporu.

- Jestliže pracujete s panelem Akce v Normálním Režimu, ujistěte se, že váš příkaz (statement) je nastaven na výraz (expression).

  Jestliže zadáváte výraz v akci a nebyl zvolen box Expression, hodnota bude předána jako řetězec. Viz „Použití operátorů pro manipulaci s hodnotami ve výrazech“.

- Ujistěte se, že žádné prvky ActionScriptu nemají stejné jméno.

  Je dobré dát každé proměnné, objektu a vlastnosti unikátní jméno. Lokální proměnné jsou výjimky: musí být unikátní pouze ve svém prostoru a jsou často používané znovu jako počítadla. Viz “Rozsah proměnné“.

Více tipů o odstraňování problémů ve Flash movie viz Flash Support Center na http://www.macromedia.com/support/flash.

Použití Odstraňovače závad (Debugger)

Odstraňovač závad umožňuje najít chyby v animaci při jejím běhu ve Flash Přehrávači. Pro určování správných hodnot si můžete prohlížet seznam zobrazení movie klipů, natažené externí animace a změny hodnot proměnných a vlastností. Můžete se pak vrátit do skriptu a editovat je tak, aby dávaly požadované správné výsledky. Při použití Odstraňovače závad musíte spustit Flash Debug Přehrávač, speciální verzi Flash Přehrávače.

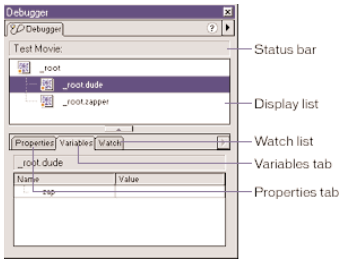
Flash Debug Přehrávač (Player) je automaticky instalován s autorskou aplikací Flash 5. Umožňuje stáhnout seznam zobrazení, jméno proměnné a dvojice hodnot, jméno vlastnosti a dvojice hodnot do Odstraňovače závad ve Flash autorské aplikaci.

Zobrazení Odstraňovače závad:
Zvolte Window > Debugger (Odstraňovač závad).

Toto otevře Odstraňovač závad v neaktivním stavu. V seznamu zobrazení se neobjeví žádná informace, dokud není vydán příkaz z Flash Přehrávače.

Aktivace Odstraňovače závad v režimu test-movie:
Zvolte Control > Debug Movie.

Toto otevře Odstraňovač závad v aktivním stavu.



Umožnění odstraňování závad v animaci

Pokud exportujete animaci, můžete zvolit možnost odstraňování závad a vytvořit heslo pro odstraňování závad. Jestliže neumožníte odstraňování závad, Odstraňovač závad se nebude aktivovat.

Stejně jako v JavaScriptu nebo HTML, jakékoliv klientské proměnné mohou být potencionálně prohlíženy uživatelem. Abyste proměnné bezpečně uložili, musíte je poslat do serverové aplikace namísto jejich ukládání do animace.

Nicméně, jako vývojář Flash, můžete mít svoje obchodní tajemství, například struktury movie klipů apod., které nechcete prozradit. Abyste zajistili, že pouze vámi vybraní důvěryhodní uživatelé si mohou vaše animace prohlížet s Flash Debug Přehrávačem, můžete vaši animaci publikovat s heslem Odstraňovače závad.

Umožnění odstraňování závad a vytvoření hesla:

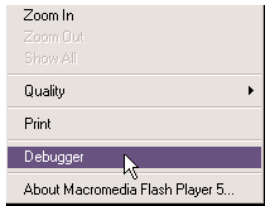
1. Zvolte File > Publish Settings.
2. Klikněte na tabulku Flash.
3. Zvolte Debugging Permitted.
4. Pro nastavení hesla vložte heslo do boxu Password.

Bez tohoto hesla nemůžete stáhnout informaci do Odstraňovače závad. Jestliže necháte pole hesla prázdné, není požadováno heslo žádné.

Aktivace Odstraňovače závad ve Web browseru:

1. Klikněte vpravo (Windows) nebo na Control (Macintosh) pro otevření kontext menu Flash Debug Player.
2. Zvolte Debugger.

Poznámka: Odstraňovač závad můžete použít pro monitorování pouze jedné animace v jednom čase. Pro použití Odstraňovače závad musí být otevřený Flash.



O sloupci stavu

Když je aktivován, Debugger status bar zobrazuje URL nebo lokální soubor cesty k animaci. Flash Přehrávač je implementován v různých formách v závislosti na prostředí přehrávání. Sloupec stavu Odstraňovače závad zobrazuje typ Flash Přehrávače, na kterém animace běží:

- Režim test-movie
- Samostatný přehrávač
- Netscape plug-in

Netscape plug-in je používán s Netscape Navigator na Windows a Macintosh a v Microsoft Internet Explorer na Macintosh.

- ActiveX kontrola
- ActiveX kontrola je používána s Internet Explorer ve Windows.

O seznamu zobrazení

Pokud je Odstraňovač závad aktivní, ukazuje živý náhled na seznam zobrazení (display list) movie klipů. Abyste viděli všechny movie klipy aktuálně na Scéně, můžete rozšířit a zúžit větve. Jestližee jsou movie klipy připojeny k nebo odstraněny z animace, seznam zobrazení ukazuje změny okamžitě. Pohybem horizontálního rozdělovače nebo tažení ze spodního pravého rohu, můžete změnit velikost seznamu zobrazení.

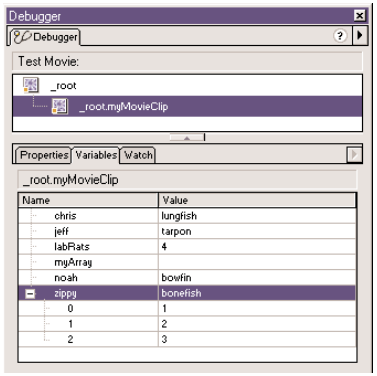
Zobrazení a modifikování proměnných

Tabulka Variable v Odstraňovači závad zobrazuje jména a hodnoty všech proměnných v animaci. Jestliže změníte hodnotu proměnné v tabulce Variable, můžete tuto změnu vidět ihned při běhu animace.

Zobrazení proměnné:

1. Zvolte movie klip obsahující proměnnou ze seznamu zobrazení.
2. Klikněte na tabulku Variable.

Seznam proměnných se automaticky aktualizuje během přehrávání animace. Jestliže je na určitém snímku movie klip odstraněn z animace, je odstraněn také ze seznamu zobrazení v Odstraňovači závad; toto odstraní jméno proměnné a také její hodnotu.



Modifikace hodnoty proměnné:
Zvolte hodnotu a vložte novou hodnotu.

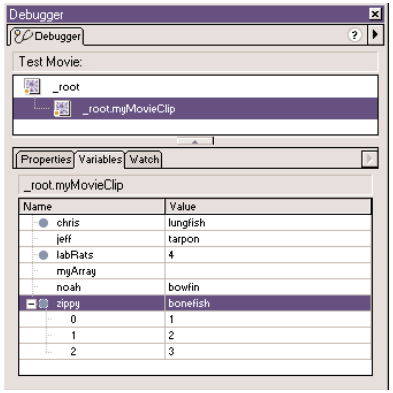
Hodnota musí být konstantní hodnota (například **"Ahoj"**, **3523**, nebo **„http://www.macromedia.com"**), ne výraz (například **x+2**, nebo **eval(„jméno:"+i)**). Hodnota může být řetězec (jakákoliv hodnota obklopená citačními znaménky (""), číslo nebo Booleánská (**true** nebo **false**).

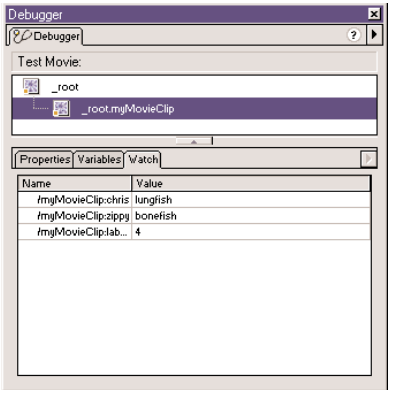
Proměnné Object a Aray jsou zobrazeny v tabulce Variables. Klikněte na tlačítko Add (**+**), abyste viděli jejich vlastnosti a hodnoty. Nicméně, vložit hodnoty Objektu nebo Pole (například **{jmeno:"Jsem objekt"}** nebo **[1,2,3]**) do hodnot polí nemůžete.

Poznámka: Pro výstup hodnoty výrazu v režimu test-movie použijte akci **trace** (stopa). Viz „Použití stopy".

Použití seznamu Watch (Sledovat)
Pro monitorování nastavení kritických hodnot můžete označit proměnné, které se mají objevit v seznamu Watch. Seznam Watch zobrazuje absolutní cestu k proměnné a hodnotě. Také zde můžete vložit novou hodnotu proměnné.

Do seznamu Watch mohou být přidány pouze proměnné, ne vlastnosti a funkce.





Pro přidání proměnných do seznamu Watch udělejte jedno z následujících:

- V tabulce Variable klikněte vpravo (Windows) nebo na Control (Macintosh) na vybranou proměnnou a zvolte Watch z kontext menu. Vedle proměnné se objeví modrá tečka.

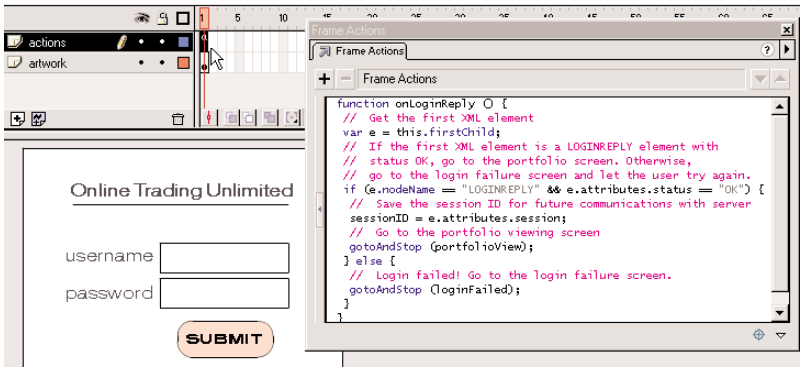
- V tabulce Watch klikněte vpravo (Windows) nebo na Control (Macintosh) a zvolte Add z kontext menu. Vložte jméno proměnné a hodnotu do textových polí.

Odstranění proměnných ze seznamu Sledovat:
V tabulce Watch klikněte vpravo (Windows) nebo na Control (Macintosh) a zvolte Remove z kontext menu.

Zobrazení vlastností animace a změna editovatelných vlastností
Tabulka Debugger Properties zobrazí všechny hodnoty vlastnosti jakéhokoliv movie klipu na Scéně. Změnit hodnotu vlastnosti a sledovat změnu můžete hned při přehrávání animace. Některé vlastnosti movie klip jsou pouze pro čtení a nemohou být měněny.

Zobrazení vlastností movie klipu:
1. Zvolte movie klip ze seznamu zobrazení.

2. Klikněte na tabulku Properties.



Modifikování hodnoty vlastnosti:
Zvolte hodnotu a vložte novou hodnotu.

Hodnota musí být konstanta (například 50 nebo **„čistávoda“**) raději než výraz (například **x+50**). Hodnota může být řetězec (jakákoliv hodnota obklopená citačními znaménky (“”), číslo nebo Booleovská hodnota (**true** nebo **false**). V Odstraňovači závad nemůžete vložit hodnoty objektu nebo pole (například **{id:“gauner“}** nebo **[1,2,3]**).

Více informací viz „Řetězec“ a „Použití operátorů pro manipulaci s hodnotami ve výrazech“.

Poznámka: Pro výstup hodnot výrazu v režimu test-movie zvolte akci stopa. Viz „Použití stopy“.

Použití okna Výstup (Output)
V režimu test-movie zobrazuje okno Output informaci, která pomáhá odstraňovat problémy v animaci. Některé informace, jako například chyby syntaxe, jsou zobrazovány automaticky. Pomocí příkazů List Objects (Seznam Objektů) a List Variables (Seznam Proměnných) si můžete nechat zobrazit další informace. Viz „Použití Seznamu Objektů“ a „Použití Seznamu Proměnných“.

Jestliže ve skriptech používáte akci **trace**, můžete poslat během přehrávání animace určité informace do okna Output. Tyto by mohly zahrnovat poznámky o stavu animace, nebo hodnotu výrazu. Viz „Použití stopy“.

Zobrazení okna Výstup:
1. Jestliže animace neběží v režimu test-movie, zvolte Control > Test Movie.

2. Zvolte Window > Output.

Objeví se okno Výstup.

Poznámka: Pokud jsou ve skriptu chyby syntaxe, okno Výstup se zobrazí automaticky.

3. Pro práci s obsahem okna Output použijte menu Options:

Zvolte Options > Copy, pro zkopírování obsahu okna Výstup do schránky (Clipboard).

Zvolte Options > Clear, pro vyčištění obsahu okna.

Zvolte Options > Save to File, pro uložení obsahu okna do textového souboru.

Zvolte Options > Print, pro tisk obsahu okna.

Použití Seznamu Objektů

V režimu test-movie zobrazuje příkaz List Objects (Seznam Objektů) úroveň, snímek, typ objektu (tvar, movie klip nebo tlačítko) a cílovou cestu movie klip instance v hierarchickém seznamu. Toto je zvláště užitečné pro hledání správné cílové cesty a jména instance. Narozdíl od Odstraňovače závad, seznam se v průběhu animace automaticky neaktualizuje; pokud chcete poslat informaci do okna Výstup, pokaždé musíte zvolit příkaz List Objects.

Zobrazení seznamu objektů v animaci:

1. Pokud animace neběží v režimu test-movie, zvolte Control > Test Movie.

2. Zvolte Debug > List Objects.

Seznam všech objektů, které jsou aktuálně na Scéně, je zobrazen v okně Výstup, jako v příkladě:

```
Layer #0:Frame=3
Movie Clip:Frame=1 Target=_root.MC
       Shape:
    Movie Clip:Frame=1 Target=_root.instance3
       Shape:
    Button:
       Movie Clip:Frame=1 Target=_root.instance3.instance2
          Shape:
```

Poznámka: Příkaz Seznam Objektů nezapisuje všechna data objektů ActionScriptu. V tomto kontextu je objekt považován za tvar nebo symbol na Scéně.

Použití Seznamu Proměnných

V režimu test-movie příkaz List Variables (Seznam Proměnných) zobrazí seznam všech proměnných, které jsou aktuálně v animaci. Toto je zvláště užitečné pro hledání správné cílové cesty k proměnné a jména proměnné. Narozdíl od Odstraňovače závad, seznam se v průběhu animace automaticky neaktualizuje; pokud chcete poslat informaci do okna Výstup, pokaždé musíte zvolit příkaz List Variables.

Zobrazení seznamu proměnných v movie:

1. Pokud animace neběží v režimu test-movie, zvolte Control > Test Movie.

2. Zvolte Debug > List Variables.

Seznam všech proměnných, které jsou aktuálně v animaci je zobrazen v okně Výstup, jako v příkladě:

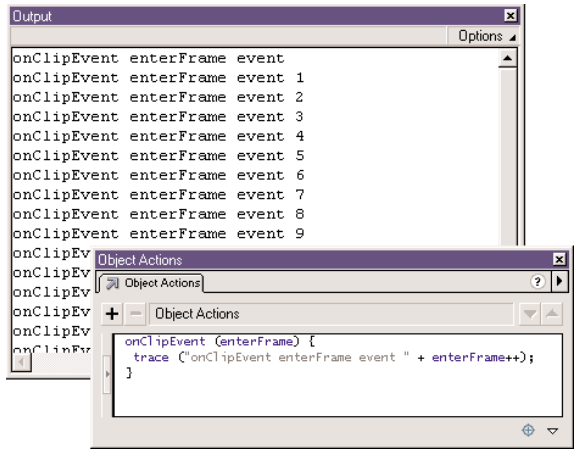
```
Level #0:
    Variable _root.country ="Sweden"
    Variable _root.city ="San Francisco"
Movie Clip:Target=""
Variable _root.instance1.firstName ="Rick"
```

Použití stopy

Když používáte akci **trace** (stopa) ve skriptu, můžete poslat informaci do okna Output. Například, zatímco testujete animaci nebo scénu, nebo pokud je stisknuto tlačítko, nebo je hrán určitý snímek, můžete poslat programovací poznámky do okna Výstup. Akce **trace** je podobná výrazu JavaScriptu **alert**.

Když používáte akci stopa ve skriptu, můžete použít výrazy jako argumenty. Hodnota výrazu je zobrazena v okně Výstup, jako v příkladu:

```
onClipEvent(enterFrame){
    trace("onClipEvent enterFrame" +enterFrame++)
}
```



KAPITOLA 7

ActionScript Slovník

Tato část **Odkazového Průvodce ActionScriptu** popisuje syntaxi a použití prvků ActionScriptu ve Flash 5 a novějších verzích. Hesla v tomto průvodci jsou stejná, jako v ActionScript Dictionary Help. Pro použití příkladů ve skriptech si zkopírujte ukázkový text z ActionScript Slovníku a vložte jej do panelu Akcí v Režimu Expert.

Ve slovníku jsou zapsány všechny prvky ActionScriptu - operátory, klíčová slova, výrazy, akce, vlastnosti, funkce, objekty a metody. Přehled o všech heslech ve slovníku najdete v Obsahu slovníku; tabulky v této části jsou dobrým startovním bodem pro hledání symbolických operátorů nebo metod, jejichž třídy objektu neznáte.

ActionScript používá ECMA-262 standard (specifikace napsaná Asociací Evropských Výrobců Počítačů - European Computer Manufacturers Association), pokud není uvedeno jinak.

V tomto slovníku jsou dva typy hesel:

- Individuální hesla pro operátory, klíčová slova, funkce, proměnné, vlastnosti, metody a výrazy

- Hesla objektu, která poskytují všeobecné detaily o předdefinovaných objektech

Použijte informace v ukázkových heslech pro interpretaci struktury a konvencí použitých v těchto dvou typech hesel.

Ukázkové heslo pro většinu prvků ActionScriptu

Následující ukázkový skript vysvětluje obecné zásady použité pro všechny ActionScript prvky, které nejsou objekty.

Název hesla

Všechna hesla jsou zapsána podle abecedy. Abecední pořádek ignoruje velká písmena, dolní pomlčky, apod.

Syntaxe

Část „Syntaxe“ poskytuje správné použití syntaxe ActionScript prvku v kódu. Část syntaxe kódu je v **`code fontu`** a argumenty, které musíte poskytnout jsou v **`italicized code fontu`**. Závorky indikují volitelné argumenty.

Argumenty

Tato část popisuje jakékoliv argumenty zapsané v syntaxi.

Popis

Tato část určuje prvek (například jako operátor, metodu, funkci nebo jiný prvek) a potom popisuje, jak je prvek používán.

Přehrávač

Tato část říká, která verze Přehrávače prvek podporuje. Toto není stejné jako verze Flashe použitá při vytváření obsahu. Například, jestliže vytváříte obsah pro Flash 4 Přehrávač pomocí použití autorských nástrojů Flash 5, nemůžete použít ty prvky ActionScriptu, které jsou dostupné pouze pro Flash 5 Přehrávač. Se zavedením Flash 5 ActionScript byly některé Flash 4 (a dřívější) ActionScript prvky zamítnuty. Ačkoliv zamítnuté prvky jsou stále podporovány Flash 5 Přehrávačem, je doporučováno, abyste používali nové prvky Flash 5.

Navíc, funkčnost operátoru ve Flash 5 byla značně rozšířena. Nejenom, že bylo zavedeno mnoho nových matematických operátorů, ale některé starší operátory jsou nyní schopné ovládat dodatečné typy dat. Pro udržení typové konzistence dat jsou Flash 4 soubory automaticky modifikovány při importu do autorského prostředí Flash 5, ale tyto modifikace neovlivňují funkčnost původního skriptu. Více informací viz hesla pro **+** (přidání), **<** (menší než), **>** (větší než), **<=** (menší nebo rovno), **>=** (větší nebo rovno), **!=** (nerovnost) a **=** (rovnost).

Příklad

Tato část poskytuje příklad kódu demonstrující použití prvku.

Viz také

V této části jsou zapsána souviseející hesla v ActionScriptu.

Ukázkové heslo pro objekty

Následující ukázkové heslo slovníku vysvětluje obecné zásady použité pro předdefinované ActionScript objekty. Objekty jsou zapsány podle abecedního pořádku se všemi dalšími prvky ve slovníku.

Název hesla

Název hesla poskytuje jméno objektu. Za jménem objektu následuje odstavec, obsahující všeobecné informace o objektu.

Shrnující tabulky metod a vlastností

Každé heslo objektu obsahuje tabulku, ve které jsou zapsány všechny metody spojené s objektem. Jestliže má objekt vlastnosti (často konstanty), tyto prvky jsou shrnuty v dodatečné tabulce. Všechny metody a vlastnosti zapsané v těchto tabulkách mají také svá vlastní hesla ve slovníku, které následují heslo objektu.

Konstruktor

Jestliže po vás objekt vyžaduje použití konstruktoru pro dosažení jeho metod a vlastností, je konstruktor popsán na konci hesla objektu. Tento popis obsahuje všechny standardní prvky (popis syntaxe atd.) dalších hesel slovníku.

Zapsání metody a vlastnosti do seznamu

Vlastnosti a metody objektu jsou zapsány abecedně za heslem objekt.

Obsahy slovníku

Všechna hesla slovníku jsou zapsána podle abecedy. Nicméně, některé operátory jsou symboly a jsou prezentovány v ASCII pořadí. Navíc, metody, které jsou spojeny s objektem jsou zapsány u jména objektu - například, metoda **abs** objektu **Math** je zapsána jako **Math.abs**.

Následující dvě tabulky vám pomohou lokalizovat tyto prvky. V první tabulce jsou zapsány symbolické operátory v pořadí, ve kterém se objeví ve slovníku. Ve druhé tabulce jsou zapsány všechny další ActionScript prvky.

Poznámka: Přednost a spojování operátorů viz Dodatek A.

Symbolické operátory	
--	(decrement)
++	(increment)
!	(logical NOT)
!=	(inequality)
%	(modulo)
%=	(modulo assignment)
&	(bitwise AND)
&&	(short-circuit AND)
&=	(bitwise AND assignment)
()	(parentheses)
-	(minus)
*	(multiplication)
*=	(multiplication assignment)
,	(comma)
.	(dot)
?:	(conditional)
/	(division)
//	(comment delimiter)

Symbolické operátory	
^	(bitwise XOR)
^=	(bitwise XOR assignment)
{ }	(object initializer)
|	(bitwise OR)
||	(logical OR)
|=	(bitwise OR assignment)
~	(bitwise NOT)
+	(addition)
+=	(addition assignment)
<	(less than)
<<	(bitwise left shift)
<<=	(bitwise left shift and assignment)
<=	(less than or equal to)
<>	(inequality)
=	(assignment)
-=	(negation assignment)
==	(equality)
>	(greater than)
>=	(greater than or equal to)
>>	(bitwise right shift)
>>=	(bitwise right shift and assignment)
>>>	(bitwise unsignet right shift)
>>>=	(bitwise unsigned right shift and assignment)

Prvek Actionscriptu	Viz heslo
abs	Math.abs
acos	Math.acos
add	add
and	and
_alpha	_alpha
appendChild	XML.appendChild
Array	Array (object)
asin	Math.asin
atan	Math.atan
atan2	Math.atan2
attachMovie	MovieClip.attachMovie
attachSound	Sound.attachSound
attributes	XML.attributes
BACKSPACE	Key.BACKSPACE
Boolean	Boolean (function, object)
break	break
call	call
CAPSLOCK	Key.CAPSLOCK
ceil	Math.ceil
charAt	String.charAT
charCodeAt	String.charCodeAt
childNodes	XML.childNodes
chr	chr
cloneNode	XML.cloneNode
close	XMLSocket.close
Color	Color (object)
concat	Array.concat,String.concat
connect	XMLSocket.Connect
constructor	Array, Boolean, Color, Date, Number, Object, Sound, String, XML, XMLSocket
continue	continue
CONTROL	Key.CONTROL
cos	Math.cos
createElement	XML.createElement
createTextNode	XML.createTextNode
_currentframe	currentframe

Prvek Actionscriptu	Viz heslo
Date	Date (object)
delete	delete
DELETEKEY	Key.DELETEKEY
docTypeDecl	XML.docTypeDecl
do...while	do...while
DOWN	Key.DOWN
_DROPTARGET	_droptarget
DUPLICATEmOVIEcLIP	duplicateMovieClip, MovieClip.duplicateMovieClip
E	Math.E
else	else
END	Key.END
ENTER	Key.ENTER
eq	eq (equal-string specific)
escape (function)	escape (function)
ESCAPE (constant)	Key.ESCAPE
eval	eval
evaluate	evaluate
exp	Match.exp
firstChild	XML.firstChild
floor	Math.floor
_focusrect	_focusrect
for	for
for...in	for...in
_framesloaded	_framesloaded
fromCharCode	String.fromCharCode
fscommand	fsconnand
function	function
ge	ge (greater than or equal to-string specific)
getAscii	Key.getAscii
getBeginIndex	Selection.getBinIndex
getBounds	MovieClip.getBounds
getBytesLoaded	MovieClip.getBytesLoaded
getBytesTotal	MovieClip.getBytesTotal
getCaretIndex	Selection.getCaretIndex
getCode	Key.getCode

Prvek Actionscriptu	Viz heslo
getDate	Date.getDate
getDay	Date.getDay
getEndIndex	Selection.getEndIndex
getFocus	Selection.getFocus
getFullZear	Date.getFullYear
getHours	Date.getHours
getMilliseconds	Date.getMilliseconds
getMinutes	Date.getMinutes
getMonth	Date.getMonth
getPan	Sound.getPan
getProperty	getProperty
getRGB	Color.getRGB
getSeconds	Date.getSeconds
getTime	Date.getTime
getTimer	getTimer
getTimezoneOfset	Date.getTimezoneOfset
getTransform	Color.getTransform, Sound.getTransform
getURL	getURL, MovieClip.getURL
getUTCDate	Date.getUTCDate
getUTCDay	Date.getUTCDay
getUTCFullYear	Date.getUTCFullYear
getUTCHours	Date.getUTCHours
getUTCMilliceconds	Date.getUTCMilliceconds
getUTCMinutes	Date.getUTCMinutes
getUTCMonth	Date.getUTCMonth
getUTCSeconds	Date.getUTCSeconds
getVersion	getVersion
getVolume	Sound.getVolume
getYear	Date.getYear
globalToLocal	MovieClip.globalToLocal
gotoAndPlay	gotoAndPlay, MovieClip.gotoAndPlay
gotoAndStop	gotoAndStop, MovieClip.gotoAndStop
gt	gt(greater than-string specific)
hasChildNodes	XML.hasChildNodes
_height	_height

Prvek Actionscriptu	Viz heslo
hide	Mouse.hide
_highquality	_highquality
hitTest	MovieClip.hitTest
HOME	Key.HOME
if	if
ifFrameLoaded	ifFrameLoaded
#include	#include
indexOf	String.indexOf
Infinity	Infinity
INSERT	Key.INSERT
insertBefore	XML.insertBefore
int	int
isDown	Key.isDown
isFinite	isFinite
isNan	isNan
isToggled	Key.isToggled
join	Array.join
Key	Key (object)
lastChild	XML.lastChild
lastIndexOf	StringlastIndexOf
le	le (less than or equal to-string specific)
LEFT	Key.LEFT
length	length, Array.length, String.length
LN2	Math.LN2
LN10	Math.LN10
load	XML.load
loaded	XML.loaded
loadMovie	loadMovie, MovieClip.loadMovie
loadVeriables	loadVeriables, MovieClip.loadVeriables
localToGlobal	MovieClip.localToGlobal
log	Math.log
LOG2E	Math.LOG2E
LOG10E	Math.LOG10E
lt	lt (less than or equal to-string specific)

Prvek Actionscriptu	Viz heslo
Math	Math (object)
max	Math.max
maxscroll	maxscroll
MAX_VALUE	Number.MAX_VALUE
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
min	Math.min
MIN_VALUE	Number.MIN_VALUE
Mouse	Mouse (object)
MovieClip	MovieClip (object)
_name	_name
NaN	NaN, Number.NaN
ne	ne (not equal-string specific)
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
new (operator)	new (operator)
newline	newline
nextFrame	nextFrame, MovieClip.nextFrame
nextScene	nextScene
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
not	not
null	null
Number	Number (function, object)
Object	Object (object)
On	On (mouseEvent)
onClipEvent	onClipEvent
onClose	XMLSocket.onClose
onConnect	XMLSocket.onConnect
OnLoad	XML.OnLoad
onXML	XMLSocket.onXML
or (logical OR)	or (logical OR)

Prvek Actionscriptu	Viz heslo
ord	ord
_parent	_parent
parentNode	XML.parentNode
parseFloat	parseFloat
parseInt	parseInt
parseXML	XML.parseXML
PGDN	Key.PGDN
PGUP	Key.PGUP
PI	Math.PI
play	play, MovieClip.play
pop	Array.pop
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow
prevframe	prevframe, MovieClip.prevframe
previousSibiling	XML.previousSibiling
prevScene	prevScene
print	print
printAsBitmap	printAsBitmap
push	Array.push
_quality	_quality
radom	radom, Math.radom
removeMovieClip	removeMovieClip, MovieClip.removeMovieClip
removeNode	XML.removeNode
return	return
reverse	Array.reverse
RIGHT	Key.RIGHT
_root	_root
_rotation	_rotation
round	Math.round
scroll	scroll
Selection	Selection (object)
send	XML.send, XMLSocket.send
sendAndLoad	XML.sendAndLoad
set	set
setDate	Date.setDate

Prvek Actionscriptu	Viz heslo
setFocus	Selection.setFocus
setFullYear	Date.setFullYear
setHours	Date.setHours
setMilliseconds	Date.setMilliseconds
setMinutes	Date.setMinutes
setMonth	Date.setMonth
setPan	Sound.setPan
setPropety	setPropety
setRGB	Color.setRGB
setSeconds	Date.setSeconds
setSelection	Selection.setSelection
setTime	Date.setTime
setTransform	Color.setTransform, Sound.setTransform
setUTCDate	Date.setUTCDate
setUTCFullYear	Date.setUTCFullYear
setUTCHours	Date.setUTCHours
setUTCMilliseconds	Date.setUTCMilliseconds
setUTCMinutes	Date.setUTCMinutes
setUTCMonth	Date.setUTCMonth
setUTCSecodnds	Date.setUTCSecodnds
setVolume	Sound.setVolume
setYear	Date.setYear
Shift (method)	Array.Shift (method)
SHIFT (constant)	Key.SHIFT (constant)
show	Mouse.show
sin	Math.sin
slice	Array.slice, String.slice
sort	Array.sort
Sound	Sound (object)
_soundbuftime	_soundbuftime
SPACE	Key.SPACE
splice	Array.splice
split	String.split
sqrt	Math.sqrt
SQRT1_2	Math.SQRT1_2

Prvek Actionscriptu	Viz heslo
SQRT2	Math.SQRT2
start	Sound.start
startDrg	startDrg, MovieClip.startDrg
status	XML.status
stop	stop, MovieClip.stop, Sound.stop
stopAllSounds	stopAllSounds
stopDrag	stopDrag, MovieClip.stopDrag
String	String (function), String (object), "„(string delimer)
substr	String.substr
substring	substring, String.substring
swapDepths	MovieClip.swapDepths
TAB	Key.TAB
tan	Math.tan
_tragen	_tragen
tragetPath	tragetPath
tellTraget	tellTraget
this	this
toggleHighQuality	toggleHighQuality
toLowerCase	String.toLowerCase
toString	Array.toString, Boolean.toString, Date.toString, Number.toString,, Object.toString, XML.toString
_totalframes	totalframes
toUpperCase	String.toUpperCase
trace	trace
typeof	typeof
unescape	unescape
unloadMovie	unloadMovie, MovieClip.unloadMovie
unshift	Array.unshift
UP	Key.UP
updateAfterEvent	updateAfterEvent
_url	_url
UTC	Date.UTC
valueOf	Boolean.valueOf, Number.valueOf, Object.valueOf
var	var
_visible	_visible
void	void

Prvek Actionscriptu	Viz heslo
while	while
_width	_width
with	with
_x	_x
XML	XML (object)
xmlDecl	xmlDecl (object)
XMLSocket	XMLSocket
_xmouse	_xmouse
_xscale	_xscale
_y	_y
-ymouse	-ymouse
_yscale	_yscale

-- (decrement)

Syntaxe

--výraz
výraz --

Argumenty

výraz Proměnná, číslo, prvek v poli nebo vlastnost objektu.

Popis

Operátor; decrement (snížení) před a po operátoru, který odečítá 1 z **výrazu**. Forma operátoru snížení před výrazem (**--výraz**), odečítá 1 z výrazu a dává výsledek. Forma operátoru snížení po výrazu (**výraz--**), odečítá 1 z výrazu a dává původní hodnotu **výrazu** (výsledek před odečítáním).

Přehrávač

Flash 4 nebo novější.

Příklad

Forma operátoru snížení před, snižuje **x** na 2 (**x-1=2**) a dává výsledek jako **y**:

```
x = 3;
y = --x
```

Forma operátoru snížení po, snižuje **x** na 2 (**x-1=2**) a dává původní hodnotu (**x=3**) jako výsledek **y**:

Jestliže
```
x = 3;
y = x--
```


++ (increment)

Syntaxe

++výraz
výraz++

Argumenty

výraz Proměnná, číslo, prvek v poli nebo vlastnost objektu.

Popis

Operátor; increment (zvýšení) před a po přidává 1 k **výrazu**. Forma operátoru před výrazem (**++výraz**), přidává 1 k výrazu a dává výsledek. Forma operátoru po výrazu (**výraz++**), přidává 1 k výrazu a dává původní hodnotu **výrazu** (výsledek před přidáním).

Forma operátoru zvýšení před, zvyšuje x na 2 (**x+1=2**) a dává výsledek jako **y**:

```
x = 1;
y = ++x
```

Forma operátoru zvýšení po, zvyšuje x na 2 (**x+1=2**) a dává původní hodnotu (**x=1**) jako výsledek **y**:

```
x = 1;
x = x++;
```

Přehrávač

Flash 4 a novější.

Příklad

Následující příklad používá **++** jako operátor zvýšení před s příkazem **while**.

```
i = 0
while ( i++ < 5 ){
//tato část se vykoná pětkrát
}
```

Následující příklad používá **++** operátor zvýšení před:

```
var a = [ ];
var i = 0;
while ( i<10 ){
       a.push( ++i );
}
trace (a.join());
```

Tento skript tiskne následující:

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

Následující příklad používá ++ jako operátor zvýšení před:

```
var a = [];
var i = 0;
while ( i<10 ){
       a.push ( i++ );
}
trace (a.join());
```

Tento scénář tiskne následující:

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

! (logical NOT)

Syntaxe

!výraz

Argumenty

výraz	Proměnná nebo ohodnocený výraz.

Popis

Operátor (logický); převrací Booleovskou hodnotu proměnné nebo **výraz**. Jestliže je výrazem proměnná s absolutní nebo konvertovanou hodnotou **true**, **!proměnná** hodnota **!výrazu** je **false**. Jestliže se výraz **x && y** ohodnocuje na false, výraz **!(x && y)** se ohodnotí na **true**. Tento operátor je identický k operátoru **not**, který byl použit ve Flash 4.

Přehrávač

Flash 4 a novější.

Příklad

V následujícím příkladu je proměnná **stastny** nastavena na **false**, podmínka **if** ohodnocuje podmínku **!stastny** a jestliže je podmínka **true**, **trace** pošle řetězec do okna Output.

```
stastny = false;
if  ( !stastny ){
trace („nedělej si starosti, buď šťastný");
}
```

Následující ilustruje výsledky ! operátoru:

!true dává **false**
!false dává **true**

!= (inequality)

Syntaxe

výraz1 != výraz2

Argumenty

výraz1, výraz2	Čísla, řetězce, Booleánské, proměnné, objekty, pole nebo funkce.

Popis

Operátor (equality); testuje přesný opak operátoru ==. Jestliže **výraz1** je roven **výrazu2**, výsledek je **false**. Jako u operátoru == závisí definice **rovnosti** na typu dat, která jsou srovnávána.

- Čísla, řetězce a Booleovské hodnoty jsou porovnávány hodnotou.
- Proměnné, objekty, pole a funkce jsou porovnávány odkazem.

Přehrávač

Flash 5 a novější.

Příklad

Následující příklad ilustruje výsledky operátoru **!=**

5 != 8 dává **true**
5 != 5 dává **false**

Následující příklad ilustruje použití operátoru **!=** v příkazu **if**:

```
a = „David"
b = „blazen"
if ( a != b ){
trace („David není blázen");
}
```

Viz také

== (equality)

% (modulo)

Syntaxe
výraz1 `%` **výraz2**

Argumenty
výraz1, výraz2 Čísla, celá čísla, plynoucí čísla nebo řetězce, které se konvertují na numerickou hodnotu.

Popis
Operátor (aritmetický); vypočítává zbytek **výrazu1** dělený **výrazem2**. Jestliže je jeden z argumentů **výrazu** nenumerický, modulo operátor přistupuje k jejich konverzi na čísla.

Přehrávač
Flash 4 nebo novější.
V souborech Flash 4 je operátor `%` rozšířen v SWF souboru jako **`x-int(x/y)*y`** a nemůže být tak rychlý nebo přesný jako implementace Flash 5 Přehrávače.

Příklad
Následující příklad je numerický příklad použití operátoru `%`:

```
12 % 5 dává 2
4.3 % 2.1 dává 0.1
```

%= (modulo assignment)

Syntaxe
výraz1 `%=` **výraz2**

Argumenty
výraz1,výraz2 Celá čísla a proměnné.

Popis
Operátor (assignment); přiřazuje **výrazu1** hodnotu **výraz1** `%` **výraz2**.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující ilustruje použití operátoru `%=` s proměnnými a čísly:
`x %= y` je stejné jako **`x = x % y`**

Jestliže **`x = 14`** a **`y = 5`**, potom **`x %= 5 dává 4`**

Viz také
`% (modulo)`

& (bitwise AND)

Syntaxe
výraz1 & výraz2

Argumenty
výraz1,výraz2 Jakékoliv číslo.

Popis
Operátor (bitwise – Bit po bitu). Označení operátoru, který zpracovává danou operaci mezi dvěma či více operandy bit po bitu, tj. nejdříve první bit prvního operandu s prvním bitem druhého operandu atd.); konvertuje **výraz1** a **výraz2** na 32-bitová neoznačená celá čísla a provádí Booleovskou (AND) operaci na každém bitu argumentů celého čísla. Výsledkem je nové 32-bitové neoznačené celé číslo.

Přehrávač
Flash 5 a novější. Ve Flash 4 byl operátor **`&`** používán pro spojování řetězců. Ve Flash 5 je operátor **`&`** bitwise AND a operátory **`add`** a **+** spojují řetězce. Soubory Flash 4, které používají operátor **`&`** jsou automaticky konvertovány na operátor **`add`**, pokud jsou přeneseny do autorského prostředí Flash 5.

&& (short-circuit AND)

Syntaxe
výraz1 && výraz2

Argumenty
výraz1,výraz2 Čísla, řetězce, proměnné nebo funkce.

Popis
Operátor (logický); vykonává Booleovskou operaci na hodnotách jednoho nebo obou výrazů. Způsobuje, že **Flash** interpret ohodnotí **výraz1** (levý výraz) a dává **false**, jestliže je **výraz** ohodnocen jako **false**. Jestliže se **výraz1** ohodnotí jako true, **výraz2** (pravý) je ohodnocen. Jestliže se **výraz2** ohodnotí jako **true**, konečný výsledek je **true**; jinak je **false**.

Přehrávač
Flash 4 a novější.

Příklad
Tento příklad určuje hodnoty ohodnocených výrazů na proměnné **vitez** a **porazeny** v pořadí, jak je prováděn test:

```
vitez = ( cokoladovaVejce >= 10 ) && (zeleFazole >= 25);
porazeny = ( cokoladovaVejce <= 1 ) && ( zeleFazole <= 5 );
if ( vitez ){
      varovat = „Vyhrál jsi lov!“;
      if ( porazeny ){
            varovat = „Nyní je to nešťastný lov!“;
      }
}else{
      varovat = „Všichni jsme vítězové!“;
}
```


&= (bitwise AND assignment)

Syntaxe
výraz1 &= výraz2

Argumenty
výraz1,výraz2 Celá čísla a proměnné.

Popis
Operátor (bitwise assignment); určuje **výrazu1** hodnotu **výraz1 & výraz2**.

Přehrávač
Flash 5 a novější.

Příklad
Následující ilustruje použití operátoru **&=** s proměnnými a čísly:
x &= y je stejné jako **x = x & y**

Jestliže **x = 15** a **y = 9** potom **x &= 9** dává **9**

Viz také
& (bitwise AND)

() (parentheses)

Syntaxe
(výraz1,výraz2);
function (functionCall1, ...,functionCallN);

Argumenty
výraz1,výraz2 Čísla, řetězce, proměnné nebo text.

function Funkce, která má být provedena na obsahu v závorkách.

functionCall1, ...,functionCallN
Série funkcí, které se mají provést před tím, než je výsledek poslán do funkce mimo závorky.

Popis

Operátor (obecný); vykonává skupinovou operaci na jednom nebo více argumentech nebo obklopuje jeden nebo více argumentů a posílá výsledky jako parametr do funkce mimo závorky.

Použití 1: Vykonává skupinovou operaci na jednom nebo více výrazech pro kontrolu pořadí vykonání operátorů ve výrazu. Tento operátor přepisuje automatický pořádek a způsobuje, že výrazy uvnitř závorek jsou ohodnoceny jako první. Jestliže jsou další vloženy závorky, Flash ohodnotí obsah nejvnitřnějších závorek a potom obsahy vnějších závorek.

Použití 2: Obklopují jeden nebo více argumentů a vkládají je jako parametry do funkce mimo závorky.

Přehrávač

Flash 4 a novější.

Příklad

(Použití 1) Následující výrazy ilustrují použití závorek k řízení pořadí vykonání výrazů. (Výsledek je pod každým výrazem.)

```
(2+3)*(4+5)
45
2+(3*(4+5))
29
2+(3*4)+5
19
```

(Použití 2) Následující příklad ilustruje použití závorek u funkce:

```
getDate();
invoice(item, amount);
```

Viz také

`with`

- (minus)

Syntaxe

(Negace) **- výraz**
(Odčítání) **výraz1 - výraz2**

Argumenty

výraz1,výraz2 Jakékoliv číslo.

Popis

Operátor (aritmetický); používán pro negaci nebo odčítání. Pokud je použit pro negaci, obrací znaménko numerického **výrazu**. Pokud je použit pro odčítání, vykonává aritmetické odčítání na dvou numerických výrazech, odečítá **výraz2** od **výrazu1**. Jestliže jsou oba výrazy celá čísla, rozdíl je celé číslo. Jestliže je jeden nebo oba výrazy „číslo s plovoucí řádkovou čárkou“, rozdíl je také „číslo s plovoucí řádkovou čárkou“.

Přehrávač

Flash 4 a novější.

Příklad

(Negace) Tento výraz obrací znaménko výrazu `2+3`:

```
-(2+3)
```

Výsledek je `-5`.

(Odčítání) Tento výraz odčítá celé číslo `2` od celého čísla `5`:

```
5-2
```

Výsledek je `3`, což je celé číslo.

(Odčítání) Tento výraz odečítá „číslo s plovoucí řádkovou čárkou“ 1.5 od „čísla s plovoucí řádkovou čárkou“ 3.25:

```
dát 3.25-1.5
```

Výsledek je `1.75`, což je „číslo s plovoucí řádkovou čárkou“.

* (multiplication)

Syntaxe
výraz1 * výraz2

Argumenty
výraz1,výraz2 Celá čísla nebo plynoucí čísla.

Popis
Operátor (aritmetický); násobí dva numerické výrazy. Jestliže jsou oba numerické výrazy celá čísla, výsledkem je celé číslo. Jestliže je jeden nebo oba výrazy „číslo s plovoucí řádkovou čárkou“, výsledkem je „číslo s plovoucí řádkovou čárkou“.

Přehrávač
Flash 4 a novější.

Příklad
Tento výraz násobí celá čísla `2` a `3`:

```
2 * 3
```

Výsledek je 6, což je celé číslo.

Příklad
Tento výraz násobí „čísla s plovoucí řádkovou čárkou“ `2.0` a `3.1416`:

```
2.0 * 3.1416
```

Výsledek je `6.2832`, což je „číslo s plovoucí řádkovou čárkou“.

*= (multiplication assignment)

Syntaxe
výraz1 *= výraz2

Argumenty
výraz1,výraz2 Celá čísla, „číslo s plovoucí řádkovou čárkou“, nebo řetězce.

Popis
Operátor (assignment); určuje **výrazu1** hodnotu **výraz1 * výraz2**.

Přehrávač
Flash 4 a novější.

Příklad
Následující ilustruje použití operátoru ***=** s proměnnými a čísly:
`x *= y` je stejné jako `x = x * y`

Jestliže `x = 5 a y = 10` potom `x *= 10` dává `50`

Viz také
`* (multiplication)`

, (comma)

Syntaxe
výraz1, výraz2

Argumenty

výraz Jakékoliv číslo, proměnná, řetězec, prvek pole nebo jiná data.

Popis
Operátor; instruuje Flash, aby ohodnotil **výraz1**, potom **výraz2** a dal hodnotu **výrazu2**. Tento operátor je používán především se smyčkovým příkazem for.

Přehrávač
Flash 4 a novější.

Příklad
Následující ukázkový kód používá operátor čárka:

```
var x = 1, b = 2, c = 3;
```

Toto je ekvivalentní s následujícím:

```
var a = 1;
var b = 2;
var c = 3;
```

. (dot operator)

Syntaxe

objekt.vlastnost_nebo_metoda
jménoinstance.proměnná
jménoinstance.dětskáinstance.proměnná

Argumenty

objekt Instance objektu. Některé objekty požadují, aby byly instance vytvořeny použitím konstruktoru pro tento objekt. Objekt může být jakýkoliv předdefinovaný ActionScript objekt nebo speciální objekt. Tento argument je vždy na levé straně operátoru tečka (.).

vlastnost_nebo_metoda Jméno vlastnosti nebo metody spojené s objektem. Všechny platné metody a vlastnosti pro předdefinované objekty jsou zapsány v souhrnných tabulkách Metody a Vlastnosti pro daný objekt. Tento argument je vždy na pravé straně operátoru tečka (.).

jménoinstance Jméno instance movie klipu.

dětskáinstance Instance movie klipu, která je dítětem hlavního movie klipu.

proměnná Proměnná v movie klipu.

Popis

Operátor; používaný pro navigaci v hierarchii movie klipu, k dosažení „uhnízděných" dětských movie klipů, proměnných nebo vlastností. Tečkový operátor je také používán pro testování nebo nastavení vlastností objektu, vykonání metod objektu nebo vytvoření struktury dat.

Přehrávač

Flash 4 a novější.

Viz také

`[ ]` (operátor přístup pole)

Příklad

Tento výraz identifikuje aktuální hodnotu proměnné vlasyBarva u movie klipu osoba:

```
osoba.vlasyBarva
```

Toto je ekvivalentní k následující Flash 4 syntaxi:

```
/osoba:vlasyBarva
```


Příklad

Následující kód ilustruje, jak může být tečkový operátor použitý pro vytvoření struktury pole:

```
ucet.jmeno = "GarySmith";
ucet.adresa = „123 HlavníTřída";
ucet.mesto = "Jakékoliv Město";
ucet.stat = „CA";
ucet.zip = „12345";
```

?: (conditional)

Syntaxe

výraz1 ? výraz2 : výraz3

Argumenty

výraz1 Výraz, který ohodnocuje na Booleánskou hodnotu, obvykle srovnávací výraz.

výraz2, výraz3 Hodnoty jakéhokoliv typu.

Popis

Operátor (podmínkový); instruuje Flash, aby ohodnotil **výraz1** a dal hodnotu **výrazu2**, jestliže je **výraz1** `true`; jinak dá hodnotu **výraz3**.

Přehrávač

Flash 4 a novější.

/ (division)

Syntaxe

výraz1 / **výraz2**

Argumenty

výraz Jakékoliv číslo.

Popis

Operátor (aritmetický); dělí **výraz1 výrazem2**. S argumenty výrazu a výsledky operace dělení se zachází a jsou vyjádřeny jako dvakrát přesnější „čísla s plovoucí řádkovou čárkou".

Přehrávač

Flash 4 a novější.

Příklad

Tento výraz dělí „číslo s plovoucí řádkovou čárkou“ `22.0` číslem `7.0` a potom zobrazí výsledek v okně Výstup:

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

Výsledek je `3.1429`, což je „číslo s plovoucí řádkovou čárkou“.

// (comment delimiter)

Syntaxe

// **komentář**

Argumenty

komentář Text, který není částí kódu a měl by být interpretem ignorován.

Popis

Komentář; indikuje začátek skriptového komentáře. Jakýkoliv text, který se objeví mezi značkami // a posledním znakem řádku je interpretován jako komentář a je ignorován interpretem ActionScriptu.

Přehrávač

Flash 1 a novější.

Příklad

Tento skript používá značky komentáře pro identifikování prvního, třetího, pátého a sedmého řádku jako komentáře:

```
//nastavit X pozici míčového movie klipu
mic = getProperty (mic._x);
//nastavit Y pozici míčového movie klipu
mic = getProperty (mic._y);
//nastavit X pozici movie klipu drak
drak = getProperty (drak._x);
//nastavit pozici Y movie klipu drak
drak_y = getProperty (drak._y);
```

Viz také

`/*  (comment delimiter)`

/* (comment delimiter)

Syntaxe

/***komentář***/

/*
komentář
komentář
*/

Argumenty

komentář Jakýkoliv text

Popis

Komentář; indikuje jeden nebo více řádků skriptového komentáře. Jakýkoliv text, který se objeví mezi otevírající značkou komentáře `/*` a zavírající značkou`*/`, je interpretován jako komentář a je ignorován ActionScript interpretem. První syntaxi používejte pro identifikaci jednořádkových komentářů a druhou syntaxi pro identifikaci komentáře na více po sobě jdoucích řádcích. Vynechání zavírací značky `*/` způsobí, že ActionScript nahlásí chybovou zprávu.

Přehrávač

Flash 5 a novější.

Viz také

`// (comment delimiter)`

/= (division assignment)

Syntaxe

výraz1 /= **výraz2**

Argumenty

výraz1,výraz2 Celá čísla, plynoucí čísla nebo řetězce.

Popis

Operátor (assignment); určuje výrazu1 hodnotu výraz1 / výraz2.

Přehrávač

Flash 4 a novější.

Příklad

Následující ilustruje použití operátoru **/=** s proměnnými a čísly:

```
x /= y je stejné jako x = x / y
x = 10;
y = 2;
x /= y;
// x nyní obsahuje hodnotu 5
```

[ ] (array access operator)

Syntaxe

myArray [“a0“,“a1“,...“aN“];
objekt [hodnota1,hodnota2,...hodnotaN];

Argumenty

myArray Jméno pole.

a0,a1,...aN Prvky v poli.

hodnota1,2,...N Jména vlastností.

Popis

Operátor; vytváří nový objekt inicializací vlastností určených v argumentech nebo inicializuje nové pole s prvky (**a0**) určenými v argumentech.

Vytvořený objekt má generický objekt Object jako svůj prototyp. Použití tohoto operátoru je stejné jako volání **new Object** a použití vlastností pomocí operátoru určení. Použití tohoto operátoru je alternativní k použití operátoru **new**, který umožňuje rychlé a vhodné vytvoření objektů.

Přehrávač

Flash 4 a novější.

Příklad

Následující příkladové kódy jsou dva různé způsoby vytvoření nového prázdného objektu Array:

```
myArray = [ ];
myArray = new Array();
```


Následující je příkladem jednoduchého pole:

```
myArray = [“red“, „yellow“, „green“, „blue“, „purple“]
myArray[0]=“red“
myArray[1]=“yellow“
myArray[2]=“green“
myArray[3]=“blue“
myArray[4]=“purple“
```

^ (bitwise XOR)

Syntaxe

výraz1 ^ výraz2

Argumenty

výraz1,výraz2 Jakékoliv číslo.

Popis

Operátor (bitwise); konvertuje **výraz1** a **výraz2** na 32-bitové neoznačené celé číslo a dává 1 do každé pozice, kde korespondující bity buď ve **výrazu1** nebo **výrazu2**, avšak ne v obou jsou 1.

Přehrávač

Flash 5 a novější.

Příklad

```
15 ^ 9 dává 6
(1111 ^ 1001 == 0110)
```

^ = (bitwise XOR assignment)

Syntaxe

výraz1 ^ = výraz2

Argumenty

výraz1,výraz2 Celá čísla a proměnné.

Popis

Operátor (složené určení); určuje **výrazu1** hodnotu **výraz1 ^ výraz2**.

Přehrávač

Flash 5 a novější.

Příklad

Následující je příkladem operace **^=** :

```
// 15 decimálně = 1111 binaricky
x = 15;
// 9 decimálně = 1001 binaricky
x ^= y;
dává
x ^ y (0110 binaricky)
```

Následující ilustruje použití operátoru **^=** s proměnnými a čísly:

```
x ^= y je stejné jako x = x ^ y
Jestliže x = 15 a y = 9 potom
15 ^= 9 dává 6
```

Viz také

^ (bitwise XOR)

{} (object initializer)

Syntaxe

objekt {jméno1: hodnota1 ,
jméno1: hodnota 2,
...
jménoN: hodnotaN};

Argumenty

objekt — Objekt pro vytvoření.

jméno 1,2,... N — Jméno vlastnosti.

hodnota 1,2,...N — Korespondující hodnota každé vlastnosti jméno.

Popis

Operátor; vytváří nový objekt a inicializuje ho s danou dvojicí vlastností jméno a hodnota. Vytvořený objekt má generický objekt Object, jako svůj prototyp. Použití tohoto operátoru je stejné jako volání **new Object** a určení dvojic vlastností pomocí použití operátoru assignment. Použití tohoto operátoru je alternativou k použití operátoru **new**, který umožňuje rychlé a vhodné vytvoření objektů.

Přehrávač

Flash 5 a novější.

Příklad

Následující kód ukazuje, jak může být vytvořen prázdný objekt pomocí použití operátoru inicializátor objektu a použití **new Object**:

```
objekt = {};
objekt = new Objekt();
```

Následující vytváří objekt ucet inicializující vlastnosti **jmeno, adresa, mesto, stat, zip** a **bilance**:

```
ucet = { jmeno: "John Smith",
       adresa: "123 Hlavní Třída",
       mesto: „Blossomville",
       stat: "California",
       zip: "12345",
       bilance: "1000" };
```

Následující příklad ukazuje, jak mohou být inicializátory pole a objektu „uhnízděny" v sobě navzájem:

```
osoba = { jmeno: "Peter Piper",
       deti: [ „Jack", „Jill", „Moe", ] };
```

Následující příklad je další způsob použití informace v předchozím příkladě, se stejnými výsledky:

```
osoba = new Osoba();
osoba.jmeno = ´John Smith´;
osoba.deti = new Array ();
osoba.deti [0] = ´Jack´;
osoba.deti [1] = ´Jill´;
osoba.deti [2] = ´Moe´;
```

Viz také

[ ] (operátor přístup pole)
new
Object (objekt)

| (bitwise OR)

Syntaxe
výraz1 | výraz2

Argumenty
výraz1,výraz2 Jakékoliv číslo.

Popis
Operátor (bitwise); konvertuje **výraz1** a **výraz2** na 32-bitová neoznačená celá čísla a dává 1 do každé pozice, kde korespondující bity buď **výrazu1** nebo **výrazu2** jsou 1.

Přehrávač
Flash 5 a novější.

Příklad
Následující je příklad operace bitwise OR. Všimněte si, že `15` je binaricky `1111`:

```
// 15 decimálně = 1111 binaricky
x = 15;
// 9 decimálně = 1001 binaricky
y = 9;
//x | y = binaricky
z = x | y;
z = 15
```

Následující je dalším způsobem vyjádření předchozího příkladu:

```
15 | 9 dává 15
(1111 | 0011 = 1111)
```


|| (OR)

Syntaxe
výraz1 || výraz2

Argumenty
výraz1,výraz2 Booleovská hodnota nebo výraz, který konvertuje na Booleánskou hodnotu.

Popis
Operátor (logický); ohodnocuje **výraz1** a **výraz2**. Výsledek je (true) jestliže, buď jeden nebo oba výrazy jsou ohodnoceny na `true`; výsledek je (false) pouze, když se oba výrazy rovnají false.

U ne-Booleovských výrazů, logický operátor OR způsobuje, že Flash ohodnotí výraz nalevo; jestliže může být konvertován na true, výsledek je true.
Jinak ohodnotí výraz napravo a výsledkem je hodnota tohoto výrazu.

Přehrávač
Flash 4 a novější.

Příklad
Následující příklad používá operátor `||` v příkazu `if`:

```
chtit = true;
potrebovat = true;
milovat = false;
if (chtit || potrebovat || milovat){
trace („dvě ze 3 nejsou špatné“);
}
```

|= (bitwise OR assignment)

Syntaxe
výraz1 |= výraz2

Argumenty
výraz1,výraz2 Celá čísla a proměnná.

Popis
Operátor (asignment); určuje **výrazu1** hodnotu **výraz1 | výraz2**.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující ilustruje použití operátoru |= s proměnnými a čísly:

```
x |= y je stejné jako x = x | y
Jestliže x = 15 a y = 9 potom
x |= 9 dává 15
```

Viz také
`| (bitwise OR)`

~ (bitwise NOT)

Syntaxe
~výraz

Argumenty
výraz Jakékoliv číslo.

Popis
Operátor (bitwise); konvertuje výraz na 32-bitové neoznačené celé číslo, které potom invertuje (převrátí) bity. Nebo, jednoduše řečeno, změní znaménko čísla a odečte 1. Operace bitwise NOT změní znaménko čísla a odečte 1.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující je numerické vysvětlení operace bitwise NOT provedené na proměnné:
`~a,` dává `-1` jestliže `a = 0, a` dává `-2` jestliže `a = 1,` tedy:
`~0=-1 a ~1=-2`

+ (addition)

Syntaxe
výraz1 + výraz2

Argumenty
výraz1,výraz2 Celá čísla, čísla, „číslo s plovoucí řádkovou čárkou“ nebo řetězce.

Popis
Operátor; sčítá numerické výrazy nebo spojuje řetězce. Jestliže je jeden výraz řetězec, všechny další výrazy jsou konvertovány na řetězce a jsou spojeny.

Jestliže jsou oba výrazy celá čísla, suma je celé číslo; jestliže je buď jeden nebo oba výrazy plynoucí čísla, suma je plynoucí číslo.

Přehrávač
Flash4; Flash 5 nebo novější. Ve Flash5 je + numerický operátor nebo spojovač řezězců v závislosti na typu dat argumentu. Ve Flash 4 je **+** pouze numerický operátor. Flash 4 soubory přenesené do autorského prostředí Flash 5 podstoupí proces konverze pro zvládnutí integrity typu dat. První příklad dole ilustruje proces konverze.

Příklad
Následující ilustruje konverzi Flash 4 souboru obsahujícího numerické srovnání kvality:
Flash 4 soubor:

```
x + y
```

Konvertovaný Flash 5 soubor:

```
Number(x)+Number(y)
```

Tento příkaz sčítá celá čísla 2 a 3 a potom zobrazí výsledné celé číslo 5 do okna Output:

```
trace (2+3);
```

Tento výraz sčítá „číslo s plovoucí řádkovou čárkou“ `2.5` a `3.25` a zobrazí výsledek `5.7500`, plynoucí číslo, do okna Output:

```
trace (2.5+3.25);
```

Tento výraz spojuje dva řetězce a zobrazí výsledek „dnes mám narozeniny“ do okna Output:

```
„dnes mám“ + „narozeniny“
```

Viz také
`add`

+= (additional assignment)

Syntaxe

výraz1 += výraz2

Argumenty

výraz1,výraz2 Celá čísla, „číslo s plovoucí řádkovou čárkou“ nebo řetězce.

Popis

Operátor (složené určení); určuje **výrazu1** hodnotu **výraz1 + výraz2**. Tento operátor také provádí spojování řetězců.

Přehrávač

Flsh 4 nebo novější.

Příklad

Tento následující příklad ilustruje numerické použití operátoru **+=**:
x += y je stejné jako **x = x+y**

Jestliže **x = 5** a **y = 10** potom **x += 10** dává **15**

Tento příklad ilustruje použití operátoru **+=** s řetězcovým výrazem:

```
x = „Jmenuji se“
x += „Mary“
```

Výsledek výše uvedeného kódu je:

```
„Jmenuji se Mary“
```

Viz také

+ (addition)

< (less than)

Syntaxe

výraz1 < výraz2

Argumenty

výraz1,výraz2 Čísla nebo řetězce.

Popis

Operátor (srovnání); srovnává dva výrazy a určuje, zda **výraz1** je menší než **výraz2** (**true**) nebo zda je **výraz1** větší nebo roven **výrazu2** (**false**). Řetězcové výrazy jsou ohodnoceny a srovnávány na základě počtu znaků v řetězci.

Přehrávač

Flash 4; Flah 5 nebo novější. Ve Flash 5 je < srovnávací operátor schopný zacházet s různými typy dat. Ve Flash 4 je numerickým operátorem. Flash 4 soubory přenesené do autorského prostředí Flash 5 podstupují proces konverze pro zvládnutí integrity typu dat. První příklad dole ilustruje proces konverze.

Příklad

Následující ilustruje konverzi Flash 4 souboru obsahujícího numerické srovnání kvality.

Flash 4 soubor:

```
x < y
```

Konvertovaný Flash 5 soubor:

```
Number(x) < Number(y)
```

Následující příklady ilustrují výsledky true a false pro čísla a řetězce:

```
3 < 10 nebo „Al“ < „Jack“ dávají true
10 < 3 nebo „Jack“ < „Al“ dávají false
```

<< (bitwise left shift)

Syntaxe
výraz1 << výraz2

Argumenty
výraz1 Číslo, řetězec nebo výraz, který bude posunut vlevo.

výraz2 Číslo, řetězec nebo výraz, který se konvertuje na celé číslo od 0 do 31.

Popis
Operátor (bitwise); konvertuje **výraz1** a **výraz2** na 32-bitová celá čísla a posunuje všechny bity ve **výrazu1** vlevo o počet míst určených celým číslem, které je výsledkem konverze **výrazu2**. Bitové pozice jsou prázdné jako výsledek této operace jsou vyplněny 0. Posunutí hodnoty vlevo o jednu pozici je ekvivalentní násobení dvěma.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad posunuje celé číslo 1 o deset bitů vlevo:

```
x = 1 << 10
```

Výsledkem této operace je **`x = 1024`**. Toto je proto, že decimální 1 se rovná 1 binaricky, 1 binaricky posunutá vlevo o 10 je 10000000000 binaricky a 10000000000 binaricky je 1024 decimálně.

Následující příklad posunuje celé číslo 7 o osm bitů vlevo:

```
x = 7 << 8
```

Výsledek této operace je **`x = 1792`**. Je tomu tak proto, že 7 decimálně se rovná 111 binaricky, 111 binaricky posunuté vlevo o 8 bitů je 11100000000 binaricky a 11100000000 binaricky je 1792 decimálně.

Viz také
`>>= (bitwise right shift and assignment)`

<<= (bitwise left shift alignment)

Syntaxe
výraz1 <<= výraz2

Argumenty
výraz1 Číslo, řetězec nebo výraz, který má být posunut vlevo.

výraz2 Číslo, řetězec nebo výraz, který konvertuje na celé číslo od 0 do 31.

Popis
Operátor (složené určení); tento operátor vykonává operaci bitový posuv vlevo a ukládá obsahy jako výsledek ve výrazu1.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující dva výrazy jsou ekvivalentní:

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

Viz také
`<< (bitwise left shift)`
`>>= (bitwise right shift and assignment)`

<= (less than or equal to)

Syntaxe
výraz1 <= výraz2

Argumenty
výraz1,výraz2 Číslo nebo řetězce.

Popis
Operátor (srovnání); srovnává dva výrazy a určuje, zda **výraz1** je menší nebo roven **výrazu2** (**`true`**) nebo zda je **výraz1** větší než **výraz2** (false).

Přehrávač
Flash4; Flash 5 nebo novější. Ve Flash 5 je **<=** operátor srovnání schopný manipulovat s různými typy dat. Ve Flash 4 je **<=** numerický operátor. Flash 4 soubory přenesené do autorského prostředí Flash 5 podstupují proces konverze pro podporu integrity typu dat. První příklad dole ilustruje proces konverze.

Příklad

Následující ilustruje konverzi Flash 4 souboru obsahujícího numerické srovnání kvality.

Flash 4 soubor:

```
x <= y
```

Konvertovaný Flash 5 soubor:

```
Number(x) <= Number(y)
```

Následující příklady ilustrují výsledky true a false pro čísla a řetězce:

```
5 <= 10 nebo „Al“ <= „Jack“ dává true
10 <= 5 nebo „Jack“ <= „Al“ dává false
```

<> (inequality)

Syntaxe

výraz1 <> výraz2

Argumenty

výraz1,výraz2 Čísla, řetězce, Booleánské, proměnné, objekty, pole nebo funkce.

Popis

Operátor (inequality); testuje přesný opak operátoru **==**. Jestliže je **výraz1** roven **výrazu2**, výsledek je false. Stejně jako u operátoru **==**, definice rovnosti závisí na typu dat, která jsou srovnávána:

Čísla, řetězce a Booleánské hodnoty jsou porovnávány hodnotou.
Proměnné, objekty, pole a funkce jsou porovnávány odkazem.

Tento operátor byl zavržen ve Flash 5 a uživatelé jsou naváděni pro používání nového operátoru **!=**.

Přehrávač

Flash 2 nebo novější.

Viz také

!= (inequality)

= (assignment)

Syntaxe

výraz1 = výraz2

Argumenty

výraz1 Proměnná, prvek pole nebo vlastnost objektu.

výraz2 Hodnota jakéhokoliv typu.

Popis

Operátor (určení); určuje typ výrazu2 (argument vpravo) na proměnnou, prvek pole nebo vlastnost ve výrazu1.

Přehrávač

Flash 4; Flash 5 nebo novější. Ve Flash 5 je = operátor určení a operátor == je používán pro ohodnocení rovnosti. Ve Flash 4 je = operátor numerické rovnosti. Flash 4 soubory přenesené do Flash 5 autorského prostředí podstupují proces konverze pro podporu integrity typu dat. První příklad dole ilustruje proces konverze.

Příklad

Následující ilustruje konverzi Flash 4 souboru obsahujícího numerické srovnání kvality.

Flash 4 soubor:

```
x = y
```

Konvertovaný Flash 5 soubor:

```
Number(x) == Number(y)
```

Následující příklad používá operátor assignment pro určení číselného typu dat na proměnnou **x**:

```
x = 5
```

Následující příklad používá operátor assignment pro určení řetězcového typu dat na proměnnou **x**:

```
x = „ahoj“
```

-= (negation assignment)

Syntaxe
výraz1-= výraz2

Argumenty
výraz1,výraz2 Celá čísla, „čísla s plovoucí řádkovou čárkou“ nebo řetězce.

Popis
Operátor (složené určení); určuje **výrazu1** hodnotu **výraz1** - **výraz2**.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující ilustruje použití operátoru **-=** s proměnnými a čísly:
`x -= y je` stejné jako **`x = x - y`**

Jestliže **`x = 5 a y = 10`** potom **`x -= 10 dává -5`**

== (equality)

Syntaxe
výraz1 == výraz2

Argumenty
výraz1,výraz2 Čísla, řetězce, Booleánské, proměnné, objekty, pole nebo funkce.

Popis
Operátor (equality); testuje rovnost dvou výrazů. Výsledek je **`true`**, jestliže jsou si výrazy rovny.

Definice rovnosti závisí na typu dat argumentu:
Čísla, řetězce a Booleánské hodnoty jsou porovnávány hodnotou a jsou považovány za rovné, jestliže mají stejnou hodnotu. Například dva řetězce jsou si rovné, jestliže mají stejný počet znaků.

Proměnné, objekty, pole a funkce jsou porovnávány odkazem. Dvě proměnné jsou si rovny, jestliže odkazují na stejný objekt, pole nebo funkci. Dvě oddělená pole nejsou nikdy považována za rovné, dokonce i když mají stejný počet prvků.

Přehrávač
Flash 5 a novější.

Příklad
Následující příklad používá operátor **==** s příkazem **`if`**:

```
a = „David“, b = „David“;
if (a == b){
trace („David je David“);
}
```

> (greather than)

Syntaxe
výraz1 > výraz2

Argumenty
výraz1,výraz2 Celá čísla, „čísla s plovoucí řádkovou čárkou“ nebo řetězce.

Popis
Operátor (srovnání); srovnává dva výrazy a určuje, zda je **výraz1** větší než **výraz2** (**`true`**), nebo zda je **výraz1** menší nebo roven **výrazu2** (false).

Přehrávač
Flash 4; Flash 5 nebo novější. Ve Flash 5 je > operátor srovnání schopný zacházet s různými typy dat. Ve Flash 4 je > numerický operátor. Flash 4 soubory přenesené do autorského prostředí Flash 5 podstupují proces konverze pro zvládnutí integrity typů dat. Příklad dole ilustruje proces konverze.

Příklad
Následující ilustruje konverzi Flash 4souboru obsahujícího numerické srovnání kvality.
Flash 4 soubor:

```
x > y
```

Konvertovaný Flash 5 soubor:

```
Number(x) > Number(y)
```

>= (greather than or equal to)

Syntaxe
výraz1 >= výraz2

Argumenty
výraz1,výraz2 Řetězce, celá čísla nebo „čísla s plovoucí řádkovou čárkou“.

Popis
Operátor (srovnání); porovnává dva výrazy a určuje, zda je **výraz1** větší než nebo roven **výrazu2** (`true`) nebo zda je **výraz1** menší než **výraz2** (`false`).

Přehrávač
Flash 4; Flash 5 nebo novější. Ve Flash 5 je >= operátor srovnání schopný manipulovat s různými typy dat. Ve Flash 4 je >= numerický operátor. Flash 4 soubory přenesené do autorského prostředí Flash 5 podstupují proces konverze pro podporu integrity typu dat. Příklad dole ilustruje proces konverze.

Příklad
Následující ilustruje konverzi Flash 4 souboru obsahujícího numerické srovnání kvality.
Flash 4 soubor:
```
x >= y
```

Konvertovaný Flash 5 soubor:
```
Number(x) > Number(y)
```

>> (bitwise right shift)

Syntaxe
výraz1 >> výraz2

Argumenty
výraz1 Číslo, řetězec nebo výraz, který má být posunut vpravo.

výraz2 Číslo, řetězec nebo výraz, který konvertuje na celé číslo od 0 do 31.

Popis
Operátor (bitwise); konvertuje **výraz1** a **výraz2** na 32-bitová celá čísla a posunuje všechny bity ve **výrazu1** vpravo o počet míst určených celým číslem, které je výsledkem konverze **výrazu2**. Bity, ze kterých se posunulo vpravo jsou vyřazeny. Pro zachování znaku původního výrazu, jsou bity nalevo vyplněny 0, jestliže nejvýznamnější bit (bit nejvzdálenější nalevo) **výrazu1** je 0 a jsou vyplněny 1, jestliže nejvýznamnější bit je 1. Posunutí hodnoty vpravo o jednu pozici je ekvivalentní dělení dvěma a vyřazení zbytku.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad konvertuje 65535 na 32-bitové celé číslo a posunuje ho o 8 bitů vpravo:
```
x = 65535 >> 8
```

Výsledek operace je:
```
x = 255
```

Je tomu tak proto, že 65535 decimálně se rovná 1111111111111111 binaricky (šestnáct 1), 1111111111111111 binaricky posunuté vpravo o 8 bitů je 11111111 binaricky a 11111111 binaricky je 255 decimálně. Nejvýznamnější bit je 0, protože celá čísla jsou 32-bitová, takže vyplněný bit je 0.

Následující příklad konvertuje -1 na 32-bitové celé číslo a posunuje ho o jeden bit vpravo:
```
x = -1 >> 1
```

Výsledek operace je:
```
x = -1
```
Je tomu tak proto, že -1 decimálně se rovná 11111111111111111111111111111111 binaricky (třicet dva 1), posunutí vpravo o jeden bit způsobí, že nejméně významný bit (bit nejvíce vpravo) bude vyřazen, a nejvýznamnější bit bude vyplněn 1. Výsledek je 11111111111111111111111111111111 binaricky (třicet dva 1), což reprezentuje 32-bitové celé číslo -1.

Viz také
```
>>= (bitwise right shift and assignment)
```

>>= (bitwise right shift and assignment)

Syntaxe

výraz1 >>= výraz2

Argumenty

výraz1 Číslo, řetězec nebo výraz, který má být posunut vlevo.

výraz2 Číslo, řetězec nebo výraz, který konvertuje na celé číslo od 0 do 31.

Popis

Operátor (složené určení); tento operátor vykonává operaci bitový posun vpravo a ukládá obsahy jako výsledek ve **výrazu1**.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující dva výrazy jsou ekvivalentní:

```
A >>= B
A = ( A >> B )
```

Následující komentovaný kód používá bitwise operátor **>>=**. Je to také příklad použití všech bitwise operátorů.

```
function convertToBinary(number)
{
var result = "";
for (var i=0; i<32; i++){
//Vyjmout nejméně významný bit použitím bitwise AND
var lsb = number & 1;
//Přidat tento bit do našeho výsledného řetězce
result = (lsb ? "1": "0") + result;
//Posunout číslo vpravo o jeden bit, vidět další bit
}number >>= 1;
return result;
}
convertToBinary = (479)
//Dává řetězec
00000000000000000000000111011111
//Řetězec nahoře je binarická reprezentace decimálního čísla 479.
```

Viz také

<< (bitwise left shift)

>>> (bitwise unsigned right shift)

Syntaxe

výraz1 >>> výraz2

Argumenty

výraz1 Číslo, řetězec nebo výraz, který má být posunut vpravo.

výraz2 Číslo, řetězec nebo výraz, který konvertuje na celé číslo od 0 do 31.

Popis

Operátor (**bitwise**); stejný jako operátor **bitwise right shift** (**>>**) kromě toho, že nezachovává znak původního výrazu, protože bity nalevo jsou vždy vyplněny 0.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad konvertuje -1 na 32-bitové celé číslo a posunuje ho o jeden bit vpravo:

```
x = -1 >>> 1
```

Výsledek operace je:

```
x = 2147483647
```

Je tomu tak proto, že -1 decimálně je binaricky 11111111111111111111111111111111 (třicet dva 1) a když posunete vpravo o jeden bit, nejméně významný bit (nejvíce vpravo) je vyřazen a nejvýznamnější bit (nejvíce vlevo) je vyplněn 0. Výsledkem je: 01111111111111111111111111111111 binaricky, což reprezentuje 32-bitové celé číslo 2147483647.

Viz také

>>= (bitwise right shift and assignment)

>>>=(bitwise unsigned right shift and assignment)

- **Syntaxe**
  výraz1 >>>= výraz2

- **Argumenty**
  výraz1 Číslo, řetězec nebo výraz, který má být posunutý vlevo.

  výraz2 Číslo, řetězec nebo výraz, který konvertuje na celé číslo od 0 do 31.

- **Popis**
  Operátor (složené určení); provádí operaci neoznačený posun vpravo a ukládá obsahy jako výsledek ve **výrazu1**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující dva výrazy jsou ekvivalentní:

```
A >>>= B
A = ( A >>> B )
```

- **Viz také**
  >>> (bitwise unsigned right shift)
  >>= (bitwise right shift and assignment)

add

- **Syntaxe**
  řetězec1 add řetězec2

- **Argumenty**
  řetězec1,2 Jakýkoliv řetězec.

- **Popis**
  Operátor; spojuje dva nebo více řetězců. Operátor add nahrazuje Flash 4 operátor &; Flash 4 soubory používající operátor & jsou automaticky konvertovány na použití operátoru **add**, pro spojení řetězců, pokud jsou přeneseny do autorského prostředí Flash 5. Nicméně, když vytváříte obsah pro Flash 5 přehrávač, operátor **add** je zavržen ve Flash 5 a je doporučováno používat operátor **+**. Jestliže vytváříte obsah pro Flash 4 nebo dřívější verze Přehrávače, použijte pro spojení řetězců operátor **add**.

- **Přehrávač**
  Flash 4 nebo novější.

- **Viz také**
  + (addition)

_alpha

- **Syntaxe**
  jménoinstance._alpha
  jménoinstance._alpha = hodnota;

- **Argumenty**
  jménoinstance Jméno movie klip instance.

  hodnota Číslo od 0 do 100 určující průhlednost **alfa**.

- **Popis**
  Vlastnost; nastavuje nebo získává alfa průhlednost (hodnota) movie klipu. Platné hodnoty jsou 0 (úplně průhledný) do 100 (zcela neprůhledný). Objekty v movie klipu s **_alpha** nastavenou na 0 jsou aktivní, i když jsou neviditelné. Například na tlačítko v movie klipu s vlastností **_alpha** nastavenou na 0, může být stále kliknuto.

- **Přehrávač**
  Flash 4 nebo novější.

- **Příklad**
  Následující výrazy nastavují vlastnost **_alpha** movie klipu pojmenovaného star na 30%, když je kliknuto na tlačítko:

```
on(release){
        setProperty(star._alpha = 30);
}
```

  nebo

```
on(release){
        star._alpha = 30;
}
```

and

Syntaxe
podmínka1 and podmínka2

Argumenty
podmínka1,podmínka2 Podmínky nebo výrazy, které se ohodnocují na **`true`** nebo **`false`**.

Popis
Operátor; provádí logickou operaci AND ve Flash 4 Přehrávači. Jestliže jsou oba výrazy ohodnoceny na **`true`**, potom je celý výraz **`true`**.

Přehrávač
Flash 4 a novější. Tento operátor byl zavržen pro Flash 5 a uživatelé jsou podporováni v používání nového operátoru &&.

Viz také
`&& (short-circuit AND)`

Array (object)
Objekt Array (Pole) umožňuje získat pole a manipulovat s nimi. Pole je objekt, jehož vlastnosti jsou identifikovány číslem reprezentujícím jeho pozici v poli. Tomuto číslu se někdy říká index. Všechna pole mají základ nula, což znamená, že první prvek v poli je [0], druhý prvek je [1], atd. V následujícím příkladě **`myArray`** obsahuje měsíce roku identifikované číslem.

```
myArray[0] = „Leden“
myArray[1] = „Únor“
myArray[2] = „Březen“
myArray[3] = „Duben“
```

Pro vytvoření objektu Pole použijte konstruktor **`new Array`**. Pro získání prvků pole použijte operátor přístup pole [ ].

Přehled metod pro objekt Array

Metoda	Popis
`concat`	Spojuje argumenty a vrací je jako nové pole.
`join`	Spojuje všechny prvky pole do řetězce.
`pop`	Odstraňuje poslední prvek pole a udá jeho hodnotu.
`push`	Přidává jeden nebo více prvků na konec pole a udá novou délku pole.
`reverse`	Na místě obrátí pořadí prvků v poli.
`shift`	Odstraňuje první prvek z pole a udává jeho hodnotu.
`slice`	Oddělí část pole a vrátí ji jako nové pole.
`sort`	Třídí pole v místě.
`splice`	Přidá a/nebo odstraní prvky z pole.
`toString`	Dává řetězcovou hodnotu reprezentující prvky v objektu pole.
`unshift`	Přidá jeden nebo více prvků na začátek pole a udá novou délku pole.

Přehled vlastností pro objekt Array

Metoda	Popis
`length`	Udává délku pole.

Konstruktor pro objekt Array

Syntaxe

```
new Array();
new Array (length);
new Array (prvek0, prvek1,prvek2,...prvekN);
```

Argumenty

length Celé číslo udávající počet prvků v poli. V případě nekonečných prvků délka určuje číslo indexu posledního prvku plus 1. Více informací viz vlastnost `Array.length`.

prvek ()...prvekN Seznam dvou nebo více libovolných hodnot. Hodnoty mohou být čísla, jména nebo jiné prvky určené v poli. První prvek v poli má vždy index nebo pozici 0.

Popis

Konstruktor; umožňuje vám dosáhnout a manipulovat s prvky v poli. Pole jsou založeny na nule a prvky jsou indexovány svými čísly pořadí. Jestliže nespecifikujete žádné argumenty, je vytvořeno pole s délkou nula.

Přehrávač

Flash 5 nebo dále.

Příklad

Následující příklad vytváří objekt new Array s původní délkou 0:

```
myArray = new Array ();
```

Následující příklad vytváří objekt new Array A-Team, s původní délkou 4:

```
A-Team = new Array („Jody", „Mary", „Marcelle", „Judy");
```

Původní prvky pole A-tym jsou následující:

```
myArray[0] = „Jody"
myArray[1] = „Mary"
myArray[2] = „Marcelle"
myArray[3] = „Judy"
```

Viz také

`Array.length`

Array.concat

Syntaxe

myArray.`concat`(hodnota0,hodnota1,...hodnotaN);

Argumenty

hodnota0,...hodnotaN Čísla, prvky nebo řetězce, které mají být spojeny v nové pole.

Popis

Metoda; spojuje prvky určené v argumentech jestliže jsou, a vytváří a vrací nové pole. Jestliže argumenty specifikují pole, jsou spojeny prvky tohoto pole dříve než pole samotné.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující kód spojuje dvě pole:

```
alfa = new Array(„a",“b",“c");
ciselne = new Array(1,2,3);
alfaCiselne = alfa.concat(ciselne);
//vytváří pole [„a",“b",“c",1,2,3]
```

Následující kód spojuje tři pole:

```
cislo1=[1,3,5];
cislo2=[2,4,6];
cislo3=[7,8,9];
cisla=cislo1.concat(cislo2,cislo3)//vytváří pole [1,3,5,2,4,6,7,8,9]
```

Array.join

- **Syntaxe**
  myArray.`join`();
  myArray.`join`(oddělovač);

- **Argumenty**
  oddělovač (separator) — Znak nebo řetězec, který odděluje prvky pole v obdrženém řetězci. Jestliže na tento argument zapomenete, jako nastavený oddělovač je použita čárka.

- **Popis**
  Metoda; konvertuje prvky v poli na řetězce, spojuje je, vkládá specifikovaný oddělovač mezi prvky a dává výsledný řetězec.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad vytváří pole se třemi prvky. Potom pole třikrát propojí: pomocí použití nastaveného oddělovače, potom čárky a mezery a potom znaménka plus.

```
a = new Array („Země“, „Měsíc“, „Slunce“)
//určuje „Země,Měsíc,Slunce“k mojeVar1
mojeVar1 = a.join();
//určuje“Země,Měsíc,Slunce“ k mojeVar2
mojeVar2 = a.join(“,“);
//určuje“Země+Měsíc+Slunce“ k moje Var3
mojeVar3 = a.join(“+“);
```

Array.length

- **Syntaxe**
  myArray.`length`;

- **Argumenty**
  Žádné

- **Popis**
  Vlastnost; obsahuje délku pole. Tato vlastnost je automaticky aktualizována, když je do pole přidán nový prvek. Během určování **`myArray[index]=hodnota;`** jestliže je index číslo a **`index + 1`** je větší než vlastnost **`length`**, vlastnost **`length`** je aktualizována na **`index + 1`**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující kód vysvětluje, jak je aktualizována vlastnost délka:

```
//původní délka je 0
myArray = new Array();
//myArray.length je updatována na 1
myArray[0] = ´a´;
// myArray.length je updatována na 2
myArray[1] = ´b´;
// myArray.length je updatována na 10
myArray[9] = ´c´;
```

Array.pop

- **Syntaxe**
  myArray.`pop()`;

- **Argumenty**
  Žádné

- **Popis**
  Metoda; odstraňuje poslední prvek z pole a udává hodnotu tohoto prvku.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující kód vytváří pole **`myPets`** obsahující čtyři prvky, potom odstraní jeho poslední prvek:

```
myPets = [„cat“, “dog“, “bird“, “fish“];
popped = myPets.pop();
```

Array.push

Syntaxe
myArray.push(hodnota,...);

Argumenty
hodnota Jedna nebo více hodnot pro připojení k poli.

Popis
Metoda; přidává jeden nebo více prvků na konec pole a udává novou délku pole.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující kód vytváří pole myPets obsahující dva prvky, potom do něj přidává dva prvky. Poté, co se kód vykoná, pushed obsahuje 4.

```
myPets = [„cat", "dog"];
pushed = myPets („bird", "fish");
```

Array.reverse

Syntaxe
myArray.reverse();

Argumenty
Žádné

Popis
Metoda; obrátí pole v místě.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující je příkladem použití metody **Array.reverse**:

```
var numbers = [1,2,3,4,5,6];
trace(numbers.join());
      numbers.reverse();
      trace(numbers.join());
```

Výstup:

```
1,2,3,4,5,6,
6,5,4,3,2,1
```


Array.shift

Syntaxe
myArray.shift();

Argumenty
Žádné.

Popis
Metoda; odstraňuje první prvek z pole a udává tento prvek.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující kód vytváří pole myPets a potom odstraní první prvek z pole:

```
myPets = [„cat", "dog", "bird", "fish"];
shifted = myPets.shift();
```

Obdržená hodnota je **cat**.

Viz také
Array.pop
Array.unshift

Array.slice

Syntaxe
myArray.slice(start,konec);

Argumenty
start Číslo udávající index začátečního bodu pro rozdělení. Jestliže je start záporné číslo, startující bod začíná na konci pole, kde je -1 poslední prvek.

konec Číslo udávající index posledního bodu pro rozdělení. Jestliže na tento argument zapomenete, rozdělení zahrne všechny prvky od začátku do konce pole. Jestliže je konec záporné číslo, konečný bod je určený od konce pole, kde -1 je poslední prvek.

Popis
Metoda; odděluje část nebo podřetězec pole a udává ho jako nové pole, bez modifikace původního pole. Obdržené pole zahrnuje prvek start a všechny prvky až do prvku konec.

Přehrávač
Flash 5 nebo novější.

Array.sort

Syntaxe

myArray.sort();
myArray.sort(funkcepořadí);

Argumenty

funkcepořadí Funkce volitelného srovnání používaná pro určení pořadí třídění. Dané argumenty A a B určené funkcí pořadí by měli vykonat třídění následovně:

- -1 jestliže se A objeví před B ve tříděné sekvenci
- 0 jestliže A=B
- 1 jestliže se A objeví po B ve tříděné sekvenci

Popis

Metoda; třídí pole v místě, bez dělání kopie. Jestliže zapomenete argument funkcepořadí, Flash třídí prvky v místě pomocí použití operátoru srovnání <.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá Array.sort bez určení argumentu **funkcepořadí**:

```
var ovoce = [„pomeranče", "jablka", "jahody", „ananas", "třešně"];
        trace (ovoce.join());
        ovoce.sort();
        trace (ovoce.join());
```


Výstup:

```
pomeranče,jablka,jahody,ananas,třešně
jablka,třešně,pomeranče,ananas,jahody
```

Následující příklad používá **array.sort** s určenou funkcí pořadí:

```
var heslo = [
        „gary:foo",
        „mike:bar",
        „john:snafu",
        „steve:yuck",
        „daniel:1234"
];
        function poradi (a,b) {
        //Vstupy, které mají být tříděny jsou ve formuláři
        //jméno:heslo
        //Třídit použitím pouze části jméno
        //položky jako klíče.
        var jmeno1 = a.split(´:´)[0];
        var jméno2 = b.split(´:´)[0];
 if (jmeno1 < jmeno2){
        return -1;
}else if (jmeno1 > jmeno2) {
        return 0;
        }
}
for (var i=0; i< heslo.length; i++){
        trace (hesla.join());
}
heslo.sort (poradi);
        trace („Tříděno");
for (var i=0; i< heslo.length; i++) {
        trace (heslo.join());
}
```

Výstup:

```
daniel:1234
gary:foo
john:snafu
mike:bar
steve:yuck
```

Array.splice

- **Syntaxe**
  myArray.`splice`(start,deleteCount,hodnota0,hodnota1...hodnotaN);

- **Argumenty**

  start Index prvku v poli, kde vkládání a/nebo rušení začíná.

  deleteCount Počet prvků, které mají být zrušeny. Tento počet zahrnuje prvek určený v argumentu start. Jestliže není specifikována žádná hodnota pro **deleteCount**, metoda zruší všechny hodnoty od prvku **start** do posledního prvku pole.

  hodnota Nula nebo více hodnot pro vložení do pole ve vkládacím bodě určeném v argumentu **start**. Tento argument je volitelný.

- **Popis**
  Metoda; přidává a / nebo odstraňuje prvky z pole. Tato metoda upravuje samotné pole, bez toho, že by dělal kopii.

- **Přehrávač**
  Flash 5 nebo novější.

Array.toString

- **Syntaxe**
  myArray.`toString`();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; dává hodnotu řetězce reprezentující prvky ve specifikovaném objektu Pole. Každý prvek v poli počínaje indexem 0 a konče indexem **myArray`.length-1`** je konvertován na spojený řetězec, oddělený čárkami.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad vytváří **`myArray`** a konvertuje ho na řetězec:

```
myArray = new Array ();
myArray [0] = 1;
myArray [1] = 2;
myArray [2] = 3;
myArray [3] = 4;
myArray [4] = 5;
trace(myArray.toString())
```

  Výstup:

```
1,2,3,4,5
```

Array.unshift

- **Syntaxe**
  myArray.`unshift`(hodnota1,hodnota2,...hodnotaN);

- **Argumenty**
  hodnota1,...hodnotaN Jedno nebo více čísel, prvků nebo proměnných, které mají být vloženy na začátek pole.

- **Popis**
  Metoda; přidává jeden nebo více prvků na začátek pole a udává novou délku pole.

- **Přehrávač**
  Flash 5 nebo novější.

Boolean (function)

- **Syntaxe**
  `Boolean`(výraz);

- **Argumenty**
  výraz Proměnná, číslo nebo řetězec, který má být konvertován na Booleovskou.

- **Popis**
  Funkce; konvertuje specifikovaný argument na Booleovskou a udává Booleovskou hodnotu.

- **Přehrávač**
  Flash 5 nebo novější.

Boolean (object)

Booleovský objekt je jednoduchý obalový objekt, se stejnou funkčností jako standardní **JavaScript Boolean objekt**. Použijte **Booleovský objekt**, pro získání prvotního typu dat nebo řetězcové reprezentace Booleovského objektu.

Přehled metod Booleovského objektu

Metoda	Popis
`toString`	Udává řetězcovou reprezentaci (**true**) nebo (**false**) Booleovského objektu.
`valueOf`	Udává prvotní typ hodnoty specifikovaného Booleovského objektu.

Konstruktor pro Booleovský objekt

- **Syntaxe**

```
new Boolean();
new Boolean (x);
```

- **Argumenty**

x Číslo, řetězec, Booleovská, objekt, movie klip nebo jiný výraz. Tento argument je volitelný.

- **Popis**

Konstruktor; vytváří instanci Booleánského objektu. Jestliže zapomenete na argument **x**, Booleovský objekt je inicializován s hodnotou **false**. Jestliže určíte **x**, metoda ohodnotí argument a udá výsledek jako Booleovskou hodnotu podle následujících pravidel:

- Pokud je **x** číslo, funkce udá **true**, jestliže se **x** nerovná 0, nebo **false**, pokud je **x** jakékoliv jiné číslo.
- Pokud je **x** Booleovská, funkce udá **x**.
- Pokud je **x** objekt nebo movie klip, funkce udá **true**, jestliže se **x** nerovná nule; jinak funkce udá **false**.
- Pokud je **x** řetězec, funkce udá **true**, jestliže **Number(x)** se nerovná 0; jinak funkce udá **false**.

Poznámka: Pro zvládnutí kompatibility s Flash 4, není manipulace řetězců Booleánským objektem ECMA-262 standard.

- **Přehrávač**

Flash 5 nebo novější.

Boolean.toString

- **Syntaxe**

```
Boolean.toString();
```

- **Argumenty**

Žádné.

- **Popis**

Metoda; udává řetězcovou reprezentaci, **true** nebo **false** Booleovského objektu.

- **Přehrávač**

Flash 5 nebo novější.

Boolean.valueOf

- **Syntaxe**

```
Boolean.valueOf();
```

- **Argumenty**

Žádné.

- **Popis**

Metoda; udává prvotní typ hodnoty specifikovaného Booleovského objektu a konvertuje Booleovský obalový objekt na jeho prvotní typ hodnoty.

- **Přehrávač**

Flash 5 nebo novější.

break

Syntaxe
```
break;
```

Argumenty
Žádné.

Popis
Akce; objeví se uvnitř smyčky **(for, for...in, do...while nebo while)**. Akce break instruuje **Flash**, aby skočil do zbytku těla smyčky, zastavil akci smyčkování a vykonal příkaz následující za příkazem smyčka. Akci **break** použijte pro přerušení série „uhnízděných“ smyček.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující příklad používá akci break pro opuštění jinak nekonečné smyčky:
```
i = 0;
while (true) {
       if (i >= 100) {
              break;
       }
       i++;
}
```

call

Syntaxe
```
call(snímek);
```

Argumenty
snímek Jméno nebo číslo snímku, který se má volat v kontextu skriptu.

Popis
Akce; přepíná kontext z aktuálního skriptu na skript připojený k volanému snímku. Pokud je ukončeno vykonávání skriptu, lokální proměnné přestanou existovat.

Přehrávač
Flash 4 nebo novější. Tato akce je zavržena ve Flash 5 a je doporučováno používat akci **function**.

Viz také
function

chr

Syntaxe
```
chr(číslo);
```

Argumenty
číslo ASCII kód číslo pro konvertování na znak.

Popis
Řetězcová funkce; konvertuje ASCII kód čísla na znaky.

Přehrávač
Flash 4 nebo novější. Tato funkce byla zavržena ve Flash 5 a je doporučováno používat metodu **String.fromCharCode**.

Příklad
Následující příklad konvertuje číslo 65 na písmeno „A“:
```
chr(65)=„A“
```

Viz také
String.fromCharCode

Color (object)
Objekt Color (Barva) vám umožňuje nastavit a získat hodnotu barvy RGB a změnu barvy movie klipů. Objekt Color je podporován Flash 5 a pozdějšími verzemi Flash Přehrávače.

Musíte použít konstruktor **new Color()** pro vytvoření instance objektu Color před tím, než voláte metody objektu Color.

Přehled metod objektu Color

Metoda	Popis
getRGB	Udává numerickou hodnotu RGB nastavenou posledním voláním **setRGB**.
getTransform	Udává informaci o transformaci nastavenou posledním voláním **setTransform**.
setRGB	Nastaví hexadecimální reprezentaci RGB hodnoty pro objekt Color.
setTransform	Nastavuje transformaci barvy pro objekt Color.

Konstruktor pro objekt Color

Syntaxe

```
new Color(cíl);
```

Argumenty

cíl Jméno movie klipu, na který je aplikována nová barva.

Popis

Konstruktor; vytváří objekt Color pro movie klip určený argumentem cíl.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad vytváří nový objekt Color nazvaný **myColor** pro animaci **mojeMovie**:

```
myColor = new Color(mojeMovie);
```

Color.getRGB

Syntaxe

```
myColor.getRGB();
```

Argumenty

Žádné.

Popis

Metoda; udává numerické hodnoty nastavené posledním volání **setRGB**.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující kód získává RGB hodnotu jako hexadecimální řetězec:

```
hodnota=(getRGB()).toString(16);
```

Viz také

Color.setRGB

Color.getTransform

Syntaxe

```
myColor.getTransform();
```

Argumenty

Žádné.

Popis

Metoda; udává hodnotu transformace nastavenou posledním voláním **setTransform**.

Přehrávač

Flash 5 nebo novější.

Viz také

Color.setTransform

Color.setRGB

Syntaxe

```
myColor.setRGB(0xRRGGBB);
```

Argumenty

0xRRGGBB Hexadecimální nebo RGB barva, která má být nastavena. RR, GG a BB sestávají se dvou hexadecimálních číslic určujících vyrovnání každé barevné komponenty.

Popis

Metoda; určuje RGB barvu pro objekt Color. Volání této metody přepisuje jakákoliv předchozí nastavení metodou **setTransform**.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad nastavuje hodnotu RGB barvy pro movie klip mojeMovie:

```
myColor = new Color(mojeMovie);
myColor.setRGB(0x993366);
```

Viz také

Color.setTransform

Color.setTransform

Syntaxe

myColor.`setTransform`(colorTransformObject);

Argumenty

colorTransformObject Objekt vytvořený konstruktorem generického objektu Object, určující hodnoty transformace barvy pro parametry. Barevně transformovaný objekt musí mít parametry **`ra, rb, ga, gb, ba, bb, aa, ab`**, které jsou vysvětleny dole.

Popis

Metoda; nastavuje informaci o barevné transformaci pro objekt Color. Argument **colorTransformObject** je objekt, který vytváříte použitím generického objektu Object s parametry určujícími procento a vyrovnání hodnot pro komponenty barvy červená (red), zelená (green), modrá (blue) a alfa (průhlednost), vložené ve formátu **0xRRGGBBAA**.

Parametry pro barevně transformovaný objekt jsou definovány následovně:
`ra` je procento pro červenou komponentu (-100 až 100).
`rb` je vyrovnání pro červenou komponentu (-255 až 255).
`ga` je procento pro zelenou komponentu (-100 až 100).
`gb` je vyrovnání pro zelenou komponentu (-255 až 255).
`ba` je procento pro modrou komponentu (-100 až 100).
`bb` je vyrovnání pro modrou komponentu (-255 až 255).
`aa` je procento pro afla (-100 až 100).
`ab` je vyrovnání pro alfa (-255 až 255).

Vytvoříte barevně transformovaný objekt následovně:

```
myColorTransform = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

Mohli byste také použít následující syntaxi:

```
myColorTransform = {ra:´50´,rb:´244´,ga:´40´,gb:´112´,ba:
´12´,bb:´90´,aa:´40´,ab:´70´}
```


Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad ukazuje proces vytvoření objektu new Color pro cílové movie vytvořením barevně transformovaného objektu s parametry definovanými nahoře pomocí použití konstruktoru Object a propuštěním barevně transformovaného objektu do objektu Color pomocí metody **`setTransform`**.

```
//Vytvořit objekt barva nazvaný myColor pro cíl mojeMovie
myColor = new Color(mojeMovie);
//Vytvořit barevně transformovaný objekt nazvaný myColorTransform
použitím
//generického objektu Object
myColorTransform = new Object;
//Nastavit hodnoty pro myColorTransform
myColorTransform = {ra:´50´,rb:´244´,ga:´40´,gb:´112´,ba:
´12´,bb:´90´,aa:´40´,ab:´70´}
//Spojit barevně transformovaný objekt s objektem Color vytvořeným
pro mojeMovie
myColor.setTransform(myColorTransform);
```

continue

Syntaxe

```
continue;
```

Argumenty

Žádné.

Popis

Akce; objeví se uvnitř několika typů smyčkových příkazů.

Ve smyčce **`while, continue`** způsobí, že Flash přeskočí zbytek těla smyčky a skočí na vrchol smyčky, kde je testována podmínka.

Ve smyčce **`do`**...**`while, continue`** způsobí, že Flash přeskočí zbytek těla smyčky a skočí na spodek smyčky, kde je testována podmínka.

Ve smyčce **for, continue** způsobí, že Flash přeskočí zbytek těla smyčky a skočí na ohodnocení výrazu za smyčkou **for**.
Ve smyčce **for…in, continue** způsobí, že Flash přeskočí zbytek těla smyčky a skočí zpět na vrchol smyčky, kde je vykonána další hodnota.

Přehrávač
Flash 4 nebo novější.

Viz také
do…while
for
for…in
while

_currentframe

Syntaxe
jménoinstance._currentframe

Argumenty
jménoinstance Jméno movie klip instance.

Popis
Vlastnost (pouze pro čtení); udává číslo snímku, na kterém je aktuálně umístěna hrací hlava v Časové ose.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující příklad používá **_currentframe** pro nasměrování animace, aby skočila o pět snímků dopředu od snímku obsahující akci:

```
gotoAndStop(_currentframe+5);
```


Date (object)

Objekt Date (Datum) vám umožňuje získat hodnoty datum a čas, relativní k univerzálnímu času (Grrenwich Středozemský Čas, nyní nazvaný Univerzální Koordinovaný Čas) nebo relativní k operačnímu systému, na kterém běží Flash Přehrávač. Pro volání metod objektu Date musíte nejprve vytvořit instanci objektu Date pomocí použití konstruktoru.

Objekt Date požaduje Flash 5 Přehrávač.

Metody objektu Date nejsou statické, avšak aplikujte je pouze na individuální instanci objektu Date, která je určena při volání metody.

Přehled metod pro objekt Date

Metoda	Popis
getDate	Udává den v měsíci určeného objektu Date v souladu s místním časem.
getDay	Udává den v měsíci pro určitý objekt Date v souladu s místním časem.
getFullYear	Udává čtyřciferný rok specifikovaného objektu Date v souladu s místním časem.
getHours	Udává hodinu specifikovaného objektu Date v souladu s místním časem.
getMilliseconds	Udává milisekundy specifikovaného objektu Date v souladu s místním časem.
getMinutes	Udává minuty specifikovaného objektu Date v souladu s místním časem.
getMonth	Udává měsíc specifikovaného objektu Date v souladu s místním časem.
getSeconds	Udává sekundy specifikovaného objektu Date v souladu s místním časem.
getTime	Udává počet milisekund od půlnoci 1.ledna 1970 univerzálního času, pro specifikovaný objekt Date.
getTimezoneOffset	Udává rozdíl v minutách mezi lokálním časem počítače a univerzálním časem.
getUTCDate	Udává den (datum) v měsíci specifikovaného objektu Date v souladu s univerzálním časem.
getUTCDay	Udává den v týdnu specifikovaného objektu Date v souladu s univerzálním časem.
getUTCFullYear	Udává čtyřciferný rok specifikovaného objektu Date v souladu s univerzálním časem.
getUTCHours	Udává hodinu specifikovaného objektu Date v souladu s univerzálním časem.
getUTCMilliseconds	Udává milisekundy určeného objektu Date v souladu s univerzálním časem.
getUTCMinutes	Udává minuty specifikovaného objektu Date v souladu s univerzálním časem.
getUTCMonth	Udává měsíc specifikovaného objektu Date v souladu s univerzálním časem.
getUTCSeconds	Udává sekundy specifikovaného objektu Date v souladu s univerzálním časem.
getYear	Udává rok specifikovaného objektu Date v souladu s místním časem.
setDate	Udává den v měsíci specifikovaného objektu Date v souladu s místním časem.
setFullYear	Nastavuje celý rok pro objekt Date v souladu s místním časem.
setHours	Nastavuje hodiny pro objekt Date v souladu s místním časem.
setMilliseconds	Nastavuje milisekundy pro objekt Date v souladu s místním časem.
setMinutes	Nastavuje minuty pro objekt Date v souladu s místním časem.

Metoda	Popis
`setMonth`	Nastavuje měsíc pro objekt Date v souladu s místním časem
`setSeconds`	Nastavuje sekundy pro objekt Date v souladu s místním časem.
`setTime`	Nastavuje datum pro specifikovaný objekt Date v milisekundách.
`setUTCDate`	Nastavuje datum specifikovaného objektu Date v souladu s univerzálním časem.
`setUTCFullYear`	Nastavuje rok specifikovaného objektu Date v souladu s univerzálním časem.
`setUTCHours`	Nastavuje hodinu specifikovaného objektu Date v souladu s univerzálním časem.
`setUTCMilliseconds`	Nastavuje milisekundy specifikovaného objektu Date v souladu s univerzálním časem.
`setUTCMinutes`	Nastavuje minutu specifikovaného objektu Date v souladu s univerzálním časem.
`setUTCMonth`	Nastavuje měsíc reprezentovaný specifikovaným objektem Date v souladu s univerzálním časem.
`setUTCSeconds`	Nastavuje sekundy specifikovaného objektu Date v souladu s univerzálním časem.
`setYear`	Nastavuje rok pro specifikovaný objekt Date v souladu s místním časem.
`toString`	Udává řetězcovou hodnotu reprezentující datum a čas uložený ve specifikovaném objektu Date.
`date.UTC`	Udává počet milisekund mezi půlnocí 1.ledna 1970 univerzálního času a určitým časem.

Konstruktor pro objekt Date

- **Syntaxe**

```
new Date();
new Date(rok [,měsíc [,datum [,hodina [,minuta [,sekunda
[,milisekunda ]]]]]]);
```

- **Argumenty**

year (rok) Hodnota od 0 do 99 indikující 1900 až 1999, jinak musí být určeny všechny čtyři číslice roku.

month (měsíc) Celé číslo od 0 (Leden) do 11 (Prosinec). Tento argument je volitelný.

date (datum) Celé číslo od 1 do 31. Tento argument je volitelný.

hour (hodina) Celé číslo od 0 (půlnoc) do 23 (11 hodin večer).

minute (minuta) Celé číslo od 0 do 59. Tento argument je volitelný.

second (sekunda) Celé číslo od 0 do 59. Tento argument je volitelný.

millisecond (milisekunda) Celé číslo od 0 do 999. Tento argument je volitelný.

- **Popis**

Objekt; konstruuje nový objekt Date udávající aktuální datum a čas.

- **Přehrávač**

Flash 5 nebo novější.

- **Příklad**

Následující příklad získává aktuální datum a čas:

```
nyni = new Date();
```

Následující příklad vytváří nový objekt Date pro Garyho narozeniny 7.srpna 1974:

```
gary_narozeniny = new Date (74,7,7);
```

Následující příklad vytváří nový objekt Date, který spojuje získané hodnoty metod objektu Date `getMonth`, `getDate` a `getFullYear` a zobrazuje je v textovém poli určeném proměnnou `datumTextovePole`.

```
myDate = new Date();
datumTextovePole = (myDate.getMonth()+"/"+myDate.getDate()+"/"
+ myDate.getFullYear());
```

Date.getDate

- **Syntaxe**

```
myDate.getDate();
```

- **Argumenty**

Žádné.

- **Popis**

Metoda; udává den v měsíci (celé číslo od 1 do 31) specifikovaného objektu Date v souladu s místním časem.

- **Přehrávač**

Flash 5 nebo novější.

Date.getDay

Syntaxe
myDate`.getDay();`

Argumenty
Žádné.

Popis
Metoda; udává den v měsíci (0 pro neděli, 1 pro pondělí atd.) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.getFullYear

Syntaxe
myDate`.getFullYear();`

Argumenty
Žádné.

Popis
Metoda; udává celý rok (čtyřciferné číslo, například 2000) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad používá konstruktor pro vytvoření objektu new Date a posílá získané hodnoty metodou **`getFullYear`** do okna Výstup:

```
myDate = new Date();
trace(myDate.getFullYear());
```


Date.getHours

Syntaxe
myDate`.getHours();`

Argumenty
Žádné.

Popis
Metoda; udává hodinu (celé číslo od 0 do 23) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.getMilliseconds

Syntaxe
myDate`.getMilliseconds();`

Argumenty
Žádné.

Popis
Metoda; udává milisekundy (celé číslo od 0 do 999) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.getMinutes

- **Syntaxe**
  myDate.`getMinutes();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává minuty (celé číslo od 0 do 59) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getMonth

- Syntaxe
  myDate.`getMonth();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává měsíc (0 pro leden, 1 pro únor atd.) specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getSeconds

- **Syntaxe**
  myDate.`getSeconds();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává sekundy (celé číslo od 0 do 59) specifického objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getTime

- **Syntaxe**
  myDate.`getTime();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává počet milisekund (celé číslo od 0 do 999) od půlnoci 1.ledna 1970 univerzálního času pro specifikovaný objekt Date. Použijte tuto metodu pro reprezentaci specifického okamžiku v čase, když porovnáváte dva nebo více časů definovaných v různých časových zónách.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getTimezoneOffset

- **Syntaxe**
  myDate.`getTimezoneOffset();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává rozdíl v minutách, mezi místním časem počítače a univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad udává rozdíl mezi místním dennísvětlo-šetřícím časem pro San Francisco a univerzálním časem. Dennísvětlo-šetřící čas je faktorován do daného výsledku, jestliže datum definované v objektu Date je v průběhu dennísvětlo-šetřícího času.

```
new Date().getTimezoneOffset ();
```

  Výsledek je následující:
  `420 (7 hodin * 60 minut / hodina = 420 minut)`

Date.getUTCDate

- **Syntaxe**
  myDate.`getUTCDate();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává den (datum) v měsíci ve specifikovaném objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCDay

- **Syntaxe**
  myDate.`getUTCDay();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává den v týdnu specifikovaného objektu Date v souladu s univerzálním časem.

Date.getUTCFullYear

- **Syntaxe**
  `myDate.getUTCFullYear();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává čtyřčíselný rok specifikovaného objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCHours

- **Syntaxe**
  myDate.getUTCHours();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává hodiny specifického objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCMilliseconds

- **Syntaxe**
  myDate.`getUTCMilliseconds();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává milisekundy specifikovaného objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCMinutes

- **Syntaxe**
  myDate`.getUTCMinutes();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává minuty specifikovaného objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCMonth

- **Syntaxe**
  myDate.`getUTCMonth();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává měsíc specifikovaného objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getUTCSeconds

- **Syntaxe**
  myDate.`getUTCSeconds();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává sekundy ve specifikovaném objektu Date v souladu s univerzálním časem.

- **Přehrávač**
  Flash 5 nebo novější.

Date.getYear

- **Syntaxe**
  myDate`.getYear();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává rok specifikovaného objektu Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač. Rok je celý rok minus 1900. Například, rok 2000 je reprezentován jako 100.

- **Přehrávač**
  Flash 5 nebo novější.

Date.setDate

Syntaxe
myDate.`setDate(datum);`

Argumenty
datum Celé číslo od 1 do 31.

Popis
Metoda; nastavuje den v měsíci pro specifikovaný objekt Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.setFullYear

Syntaxe
myDate.`setFullYear(rok [, měsíc [, datum ] ] );`

Argumenty
rok Čtyřciferné číslo určující rok. Dvouciferná čísla nereprezentují roky; například 99 není rok 1999, ale rok 99.

měsíc Celé číslo od 0 (leden) do 11 (prosinec). Tento argument je volitelný.

datum Číslo od 1 do 31. Tento argument je volitelný.

Popis
Metoda; nastavuje rok specifikovaného objektu Date v souladu s místním časem. Jestliže jsou specifikovány argumenty měsíc a datum, jsou nastaveny také na místní čas. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač. Výsledky getUTCDay a getDay se mohou změnit výsledkem volání této metody.

Přehrávač
Flash 5 nebo novější.

Date.setHours

Syntaxe
myDate.`setHours(hodina);`

Argumenty
hodina Celé číslo od 0 (půlnoc) do 23 (11 hodin večer).

Popis
Metoda; nastavuje hodiny pro specifikovaný objekt Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.setMilliseconds

Syntaxe
myDate.`setMilliseconds(milisekunda);`

Argumenty
milisekunda Celé číslo od 0 do 999.

Popis
Metoda; nastavuje milisekundy pro specifikovaný objekt Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač
Flash 5 nebo novější.

Date.setMinutes

Syntaxe

myDate.setMinutes(minuta);

Argumenty

minuta Celé číslo od 0 do 59.

Popis

Metoda; nastavuje minuty pro specifikovaný objekt Date v souladu s místním časem. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač

Flash 5 nebo novější.

Date.setMonth

Syntaxe

myDate.setMonth(měsíc [, datum]);

Argumenty

měsíc Celé číslo od 0 (leden) do 11 (prosinec).

datum Celé číslo od 1 do 31. Tento argument je volitelný.

Popis

Metoda; nastavuje měsíc pro specifikovaný objekt Date v místním čase. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač

Flash 5 nebo novější.

Date.setSeconds

Syntaxe

myDate.setSeconds(sekunda);

Argumenty

sekunda Celé číslo od 0 do 59.

Popis

Metoda; nastavuje sekundy pro specifikovaný objekt Date v místním čase. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

Přehrávač

Flash 5 nebo novější.

Date.setTime

Syntaxe

myDate.setTime(milisekunda);

Argumenty

milisekunda Celé číslo od 0 do 999.

Popis

Metoda; nastavuje Date pro specifikovaný objekt datum v milisekundách.

Přehrávač

Flash 5 nebo novější.

Date.setUTCDate

Syntaxe

myDate.`setUTCDate(datum);`

Argumenty

datum Celé číslo od 1 do 31.

Popis

Metoda; nastavuje datum pro specifikovaný objekt Date v univerzálním čase. Volání této metody nemodifikuje ostatní pole specifikovaného objektu Date, ale metody **`getUTCDay`** a **`getDay`** mohou udávat novou hodnotu, jestliže se změní den v týdnu jako výsledek volání této metody.

Přehrávač

Flash 5 nebo novější.

Date.setUTCFullYear

Syntaxe

myDate.`setUTCFullYear(rok [,měsíc [,datum ]]);`

Argumenty

rok Rok určený jako čtyřciferný rok, například 2000.

měsíc Celé číslo od 0 (leden) do 11 (prosinec). Tento argument je volitelný.

datum Celé číslo od 1 do 31. Tento argument je volitelný.

Popis

Metoda; nastavuje rok nebo specifikovaný objekt Date (**myDate**) v univerzálním čase. Volitelně může tato metoda nastavit také měsíc a datum reprezentovaný specifikovaným objektem Date. Žádná další pole objektu Date nejsou modifikována. Volání **`setUTCFullYear`** může způsobit, že **`getUTCDay`** a **`getDay`** budou hlásit novou hodnotu, pokud se změní den v týdnu jako výsledek této operace.

Přehrávač

Flash 5 nebo novější.

Date.setUTCHours

Syntaxe

myDate.setUTCHours(hodina [,minuta [,sekunda [,milisekunda]]]);

Argumenty

hodina Celé číslo od 0 (půlnoc) do 23 (11 hodin večer).

minuta Celé číslo od 0 do 59. Tento argument je volitelný.

sekunda Celé číslo od 0 do 59. Tento argument je volitelný.

milisekunda Celé číslo od 0 do 999. Tento argument je volitelný.

Popis

Metoda; nastavuje hodinu pro specifikovaný objekt Date v univerzálním čase.

Přehrávač

Flash 5 nebo novější.

Date.setUTCMilliseconds

Syntaxe

myDate.`setUTCMilliseconds(milisekunda);`

Argumenty

milisekunda Celé číslo od 0 do 999.

Popis

Metoda; nastavuje milisekundy pro specifikovaný objekt Date v univerzálním čase.

Přehrávač

Flash 5 nebo novější.

Date.setUTCMinutes

- **Syntaxe**
  myDate.setUTCMinutes(minuta [,sekunda [,milisekunda]]);

- **Argumenty**
  minuta Celé číslo od 0 do 59.

  sekunda Celé číslo od 0 do 59. Tento argument je volitelný.

  milisekunda Celé číslo od 0 do 999. Tento argument je volitelný.

- **Popis**
  Metoda; nastavuje minutu pro specifikovaný objekt Date v univerzálním čase.

- **Přehrávač**
  Flash 5 nebo novější.

Date.setUTCMonth

- **Syntaxe**
  myDate.setUTCMonth(měsíc [,datum]);

- **Argumenty**
  měsíc Celé číslo od 0 (leden) do 11 (prosinec).

  datum Celé číslo od 1 do 31. Tento argument je volitelný.

- **Popis**
  Metoda; nastavuje měsíc a volitelně den (datum) pro specifikovaný objekt Date v univerzálním čase. Volání této metody nemodifikuje další pole specifikovaného objektu Date, ale metody **getUTCDay** a **getDay** mohou hlásit novou hodnotu, jestliže se změní den v týdnu jako výsledek specifikace argumentu datum, při volání **setUTCMonth.**

- **Přehrávač**
  Flash 5 nebo novější.

Date.setUTCSeconds

- **Syntaxe**
  myDate.setUTCSeconds(sekunda [, milisekunda]);

- **Argumenty**
  sekunda Celé číslo od 0 do 59.

  milisekunda Celé číslo od 0 do 999. Tento argument je volitelný.

- **Popis**
  Metoda; nastavuje sekundy pro specifikovaný objekt Date v univerzálním čase.

- **Přehrávač**
  Flash 5 nebo novější.

Date.setYear

- **Syntaxe**
  myDate.setYear(rok);

- **Argumenty**
  rok Čtyřciferné číslo, například 2000.

- **Popis**
  Metoda; nastavuje rok pro specifikovaný objekt Date v místním čase. Místní čas je určen operačním systémem, na kterém běží Flash Přehrávač.

- **Přehrávač**
  Flash 5 nebo novější.

Date.toString

- **Syntaxe**
  myDate.toString();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává řetězcovou hodnotu pro specifikovaný objekt Date ve formátu pro čtení.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad dává informaci v objektu Date datumNarozeni jako řetězec:

```
var datumNarozeni = new Date(74,7,7,18,15);
trace (datumNarozeni.toString());
```

Výstup (pro Pacifický Standardní Čas):

```
Wed (Středa) Aug (Srpen) 7 18:15:00 GMT-0700 1974
```

Date.UTC

- **Syntaxe**
  Date.UTC(rok,měsíc [,datum [,hodina [,minuta [,sekunda [, milisekunda]]]]]);

- **Argumenty**

rok	Čtyčciferné číslo, například 2000.
měsíc	Celé číslo od 0 (leden) do 11 (prosinec).
datum	Celé číslo od 1 do 31. Tento argument je volitelný.
hodina	Celé číslo od 0 (půlnoc) do 23 (11 hodin večer).
minuta	Celé číslo od 0 do 59.Tento argument je volitelný.
sekunda	Celé číslo od 0 do 59. Tento argument je volitelný.
milisekunda	Celé číslo od 0 do 999. Tento argument je volitelný.

- **Popis**
  Metoda; udává počet milisekund mezi půlnocí 1.ledna 1970 univerzálního času a časem určeným v argumentech. Toto je statická metoda, která je vyvolána pomocí konstruktoru objektu Date, ne pomocí specifického objektu Date. Tato metoda umožňuje vytvořit objekt Date, který předpokládá univerzální čas, zatímco konstruktor Date předpokládá místní čas.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad vytváří objekt new Date **gary_narozeniny**, definovaný v univerzálním čase. Toto je variace příkladu univerzálního času použitého pro konstrukční metodu **new Date ()**:

```
gary_narozeniny = new Date(Date.UTC(1974,7,8));
```

delete

- **Syntaxe**
- **delete (odkaz);**

- **Argumenty**

odkaz	Jméno proměnné nebo objektu, který má být eliminován.

- **Popis**
  Operátor; ruší objekt nebo proměnnou specifikovanou jako **odkaz** a udává **true**, jestliže byl objekt úspěšně odstraněn; jinak udává **false**. Tento operátor je užitečný pro uvolňování paměti používané skripty, ačkoliv je **delete** operátor, je typicky používán jako příkaz:

```
delete x;
```

  Operátor delete nemusí uspět a může udat false, jestliže odkaz neexistuje nebo nemůže být zrušen. Předdefinované objekty a vlastnosti a proměnné deklarované pomocí var nemohou být zrušeny.

- **Přehrávač**
  Flash 5 nebo novější.

Příklad

Následující příklad vytváří objekt, používá ho a potom ho ruší, když už není více potřeba:

```
account = new Object();
account.name = ´Jon´;
account.balance = 10000;
...
delete.account;
```

Následující příklad ruší vlastnost objektu:

```
//vytvořit nový objekt „account"
account = new Object();
//určit vlastnost name k account
account.name = ´Jon´;
//zrušit vlastnost
delete account.name;
```

Následující je dalším příkladem zrušení vlastnosti objektu:

```
//vytvořit objekt Array s délkou 0
array = new Array();
//Array.length je nyní 1
array[0] = „abc";
//přidat další prvek do array, Array.length je nyní 2
array[1] = „def";
//přidat další prvek do array, Array.length je nyní 3
array[2] = „ghi";
// array[2] je zrušeno, ale Array.length se nezmění,
delete array[2];
```

Následující příklad ilustruje chování delete na odkazech na objekty:

```
//vytvořit nový objekt a připojit proměnnou ref 1 pro odkaz na objekt
ref1 = new Object();
ref1.jmeno = „jody";
//kopírovat proměnnou odkazu do nové proměnné a zrušit ref 1
ref2 = ref1;
delete ref1;
```

Jestliže by **ref1** nebyla zkopírována do **ref2**, objekt by nebyl zrušen, když jsme zrušili **ref1**, protože by zde na něj nebyly žádné odkazy. Jestliže bychom zrušili **ref2**, nebyly by zde už žádné odkazy na objekt a ten by byl zrušen a paměť, kterou využíval by byla uvolněna.

Viz také

var

do...while

Syntaxe

```
do {
příkaz;
}while (podmínka);
```

Argumenty

podmínka Podmínka pro ohodnocení.

příkaz Příkaz, který se má vykonat, jakmile je podmínka ohodnocena na **true**.

Popis

Akce; vykonává příkazy a potom ohodnocuje podmínku ve smyčce, dokud není podmínka pravda.

Přehrávač

Flash 4 nebo novější.

Viz také

break
continue

_droptarget

Syntaxe

tažitelnáInstanceJméno`._droptarget`

Argument

tažitelnáInstanceJméno Jméno movie klip instance, která byla cílem akce **startDrag**.

Popis

Vlastnost (pouze pro čtení); udává absolutní cestu ve slash (závorkové) syntaxi movie klip instance, na kterou byla puštěna **tažitelnáInstanceJméno**. Vlastnost **_droptarget** vždy udává cestu, která začíná /. Pro porovnání vlastnosti instance **_droptarget** s odkazem použijte **eval**, pro konverzi obdržené hodnoty z flash syntaxe na odkaz.

Přehrávač

Flash 4 nebo novější.

Příklad

Následující příklad ohodnocuje vlastnost **_droptarget** movie klip instance **smeti** a používá **eval** pro její konverzi ze slash syntaxe na odkaz v dot (tečkové) syntaxi. Odkaz **smeti** je potom porovnáván s odkazem na movie klip instanci odpadky. Jestliže jsou dva odkazy ekvivalentní, viditelnost smeti je nastavena na **false**. Jestliže nejsou ekvivalentní, instance **smeti** je resetována na svoji původní pozici.

```
if (eval (smeti._droptarget) == _root.odpadky){
       smeti._visible = false;
}else{
       smeti._x = x_poz;
       smeti._y = y_poz;
}
```

Proměnné **x_poz** a **y_poz** jsou nastaveny na snímku 1 movie, s následujícím skriptem:

```
x_poz = smeti._x;
y_poz = smeti._y;
```

Viz také

startDrag

duplicateMovieClip

Syntaxe

duplicateMovieClip(cíl, novéjméno, hloubka);

Argumenty

target

(cíl) Cílová cesta movie klipu, který se má duplikovat.

newname

novéjméno) Unikátní identifikátor pro duplikovaný movie klip.

depth (hloubka) Úroveň hloubky movie klipu. Úroveň hloubky je pořadí navržení, které určuje, jak se movie klipy a další objekty objeví, pokud se překrývají. První movie klip, který vytvoříte nebo instance, kterou táhnete na Scénu, je připojen na hloubku úrovně 0. Každý následný nebo duplikovaný movie klip musíte připojit k různé úrovni, abyste zabránili nahrazování movies na obsazených úrovních nebo původního movie klipu.

Popis

Akce; vytváří instanci movie klipu, zatímco movie hraje. Duplikované movie klipy vždy začínají na snímku 1 a nezáleží na tom, na kterém snímku byl původní movie klip. Proměnné v rodičovském movie klipu nejsou kopírovány do duplikovaného movie klipu. Jestliže je rodičovský movie klip zrušen, duplikovaný movie klip je také zrušen. Použijte akci **removeMovieClip**, nebo metodu, pro zrušení movie klip instance vytvořené pomocí **duplicateMovieClip**.

Přehrávač

Flash 4 nebo novější.

Příklad

Tento příkaz duplikuje movie klip instanci kvetina desetkrát. Proměnná i je použita pro vytvoření nového jména instance a hloubku.

```
on(release) {
       mnozstvi = 10;
       while(mnozstvi>0){
              duplicateMovieClip (_root.kvetina, "mc"+i, i);
              setProperty(„mc"+i, _x,random(275));
              setProperty(„mc"+i, _y, random (275));
              setProperty(„mc"+i, _alpha, random (275));
              setProperty(„mc"+i, _xscale, random (50));
              setProperty(„mc"+i, _yscale, random (50));
              i = i + 1;
              mnozstvi = mnozstvi - 1;
       }
}
```

Viz také

removeMovieClip
MovieClip.removeMovieClip

else

Syntaxe
`else { příkaz(y) };`

Argumenty
příkaz(y) Alternativní série příkazů, které mají běžet, jestliže je podmínka specifikovaná v příkazu **`if`**, false.

Popis
Akce; specifikuje akce, věty, argumenty nebo jiné podmínky, které mají běžet, jestliže je původní příkaz **`if`**, false.

Přehrávač
Flash 4 nebo novější.

Viz také
`if`

eq (equal-string specific)

Syntaxe
výraz1 `eq` výraz2

Argumenty
výraz1,výraz2 Čísla, řetězce nebo proměnné.

Popis
Srovnávací operátor; porovnává dva výrazy, zda jsou si rovny a udává **`true`**, jestliže je **výraz1** roven **výrazu2**; jinak udává **`false`**.

Přehrávač
Flash 1 nebo novější. Tento operátor byl zavržen ve Flash5; je doporučeno používat nový operátor rovnosti ==.

Viz také
`== (equality)`

escape

Syntaxe
`escape(výraz);`

Argumenty
výraz Výraz, který se má konvertovat na řetězec a zakódovat do URL-kódovacího formátu.

Popis
Funkce; konvertuje argument na řetězec a kóduje ho v URL-kódovacím formátu, kde jsou všechny alfanumerické znaky nahrazeny s `%` hexadecimálními sekvencemi.

Přehrávač
Flash 5 nebo novější.

Příklad
```
escape(„Hello{[World]}“);
```

Výsledek výše uvedeného kódu je následující:
```
Hello%7B%5BWorld%5D%7D
```

Viz také
`unescape`

eval

Syntaxe

ev**al(výraz);**

Argumenty

výraz Řetězec obsahující jméno proměnné, vlastnosti, objektu nebo movie klipu, který se má získat.

Popis

Funkce; dosahuje proměnných, vlastností, objektů nebo movie klipů jménem. Jestliže je výraz proměnná nebo vlastnost, je získána hodnota proměnné nebo vlastnosti. Jestliže je výraz objekt nebo movie klip, je získán odkaz na objekt nebo movie klip. Jestliže nemůže být prvek pojmenovaný ve výrazu nalezen, je získáno nedefinováno.

Ve Flash 4 byla funkce **eval** používána pro simulaci polí. Ve Flash 5 je doporučováno používat objekt Array, pro vytvoření polí.

Poznámka: ActionScript akce **eval** není stejná jako JavaScript funkce eval a nemůže být používána pro ohodnocení příkazů.

Přehrávač

Flash 5 nebo novější pro plnou funkčnost. Když exportujete do Flash 4 Přehrávače můžete použít **eval**, ale musíte použít slash značky a můžete dosáhnout pouze proměnných ne vlastností nebo objektů.

Příklad

Následující příklad používá **eval** pro určení hodnoty proměnné **x** a nastavuje ji na hodnotu **y**:

```
x = 3;
y = eval(„x");
```

Následující příklad používá **eval** pro odkaz na movie klip objekt spojený s movie klip instancí Mic na Scéně:

```
eval(„_root.Mic");
```

Viz také

Array (object)

evaluate

Syntaxe

```
příkaz;
```

Argumenty

Žádné.

Popis

Akce; vytváří nový prázdný řádek a vkládá; pro vložení unikátních scénářových zpráv pomocí použití pole Výraz v panelu Akce. Příkaz **evaluate** také umožňuje uživatelům, kteří skriptují ve Flash 5 Normálním Režimu panelu Akce volat funkce.

Přehrávač

Flash 5 nebo novější.

_focusrect

Syntaxe

```
_focusrect = Booleovská ;
```

Argumenty

Booleovská **true** nebo **false**.

Popis

Vlastnost (všeobecná); určuje, zda se objeví žlutý obdélník okolo tlačítka, které má aktuální střed. Nastavená hodnota **true** (nenula) zobrazí žlutý obdélník okolo aktuálně zaměřeného tlačítka nebo textového pole, když uživatel stiskne klávesu Tab pro navigaci. Specifikujte, aby se zobrazilo **false** pouze při stavu tlačítka „over" (jestliže je nějaký definován), když uživatelé navigují.

Přehrávač

Flash 4 nebo novější.

for

Syntaxe

```
for(init;condition;next);{
příkaz;
}
```

Argumenty

init — Výraz, který se má ohodnotit dříve, než začne smyčková sekvence, typicky určující výraz. Příkaz var je také povolen pro tento argument.

condition (podmínka) — Výraz, který se ohodnocuje na **true** nebo false. Podmínka je ohodnocena před každým opakováním smyčky; smyčka existuje, když je podmínka ohodnocena na false.

next (další) — Výraz, který se má ohodnotit po každém opakování smyčky; obvykle určující výraz používající operátory **++**(zvýšení) nebo **--**(snížení).

příkaz — Příkaz uvnitř těla smyčky, který se má vykonat.

Popis

Akce; smyčková konstrukce, která ohodnocuje jednou výraz **init** a potom začíná smyčková sekvence, kterou se, pokud se podmínka ohodnotí na **true**, vykoná příkaz a je ohodnocen další výraz.

Některé vlastnosti nemohou být vyjmenovány akcemi **for** nebo **for...in**. Například metody objektu **Array (Array.sort a Array.reverse)** nejsou zahrnuty ve vyjmenováních objektu **Array** a movie klip vlastnosti, jako **_x** a **_y**, nejsou vyjmenovány.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá **for** pro přidání prvků do pole:

```
for(i=0; i<10; i++){
pole[i] = (i+5)*10;
}
```

Výsledkem je následující pole:

```
[50,60,70,80,90,100,110,120,130,140]
```


Následující je příkladem použití **for** pro vykonání stejné akce opakovaně. V kódu dole smyčka **for** přidává čísla od 1 do 100:

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

Viz také

++
--
for…**in**
var

for...in

Syntaxe

```
for(proměnnáopakování in objektu){
příkaz;
}
```

Argumenty

proměnnáopakování — Jméno proměnné, které má vystupovat jako opakovaná, odkazující na každou vlastnost objektu nebo prvku v poli.

objekt — Jméno objektu, na kterém má být opakováno.

příkaz — Příkaz, který se má vykonat pro každé opakování.

Popis

Akce; smyčkuje skrz vlastnosti objektu nebo prvku v poli a vykonává příkaz pro každou vlastnost objektu.

Některé vlastnosti nemohou být vyjmenovány akcemi **for** nebo **for...in**. Například zabudované metody objektu Array (**Array.sort a Array.reverse**) nejsou zahrnuty ve vyjmenování objektu Array a movie klip vlastnosti jako **_x** a **_x** nejsou vyjmenovány.

Konstrukce **for...in** se opakuje na vlastnostech objektů v opakovaném řetězu prototypu objektu. Jestliže je dětský prototyp rodič, opakování na vlastnostech dítěte s **for...in** se bude také opakovat na vlastnostech rodiče.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující je příkladem použití **for...in** pro opakování na vlastnostech objektu:

```
mujObject= {jmeno:´Tara´,vek:27,mesto:Śan Francisco´};
for(jmeno in mujObject){
       trace(„mujObject.“+jmeno+“=„+mujObject [jmeno]);
}
```

Výstup tohoto příkladu je následující:

```
mujObject.jmeno=Tara
mujObject.vek=27
mujObject.mesto=San Francisco
```

Následující je příkladem použití operátoru typeof s **for…in** pro opakování na konkrétním typu dítěte:

```
for (jmeno in mujMovieKlip){
       if (typeof (mujMovieKlip [jmeno]) = „movieklip“){
              trace(„Mám movie klip dítě pojmenované
„+jméno);
       }
}
```

Následující příklad vyjmenovává děti movie klipu a posílá každé do snímku 2 v jejich příslušných Časových osách. Movie klip RadioTlacitkoSkupina je rodič s několika dětmi, **_CerveneRadioTlacitko_**,**_ZeleneRadioTlacitko_** a **_ModreRadioTlacitko**.

```
for (var jmeno in RadioTlacitkoSkupina){
       RadioTlacitkoSkupina [jmeno].gotoAndStop(2);
}
```


_framesloaded

Syntaxe

```
instancejméno._framesloaded
```

Argumenty

instancejméno Jméno movie klip instance, která má být ohodnocena.

Popis

Vlastnost (pouze pro čtení); počet snímků, které byly nataženy z tekoucího movie. Tato vlastnost je užitečná pro určení, zda obsahy určitého snímku, a všechny snímky před ním, byly nataženy a jsou místně dostupné v uživatelově browseru. Tato vlastnost je užitečná pro monitorování procesu stahování rozsáhlých animací. Například můžete chtít zobrazit zprávu pro uživatele, která indikuje natahování movie, dokud určitý snímek v animaci nedokončil natahování.

Přehrávač

Flash 4 nebo novější.

Příklad

Následující je příkladem použití vlastnosti **_framesloaded** pro koordinaci začátku movie s počtem natažených snímků:

```
if (_framesloaded >= _totalframes){
gotoAndPlay („Scene 1,“zacatek“);
}else{
setProperty
(„_root.natahovac“, _xscale,(_framesloaded/_totalframes)*100);
}
```

fscommand

Syntaxe

```
fscommand(příkaz,argumenty);
```

Argumenty

příkaz Řetězec propuštěný do hostitelské aplikace pro jakékoliv použití.

argumenty Řetězec propuštěný do hostitelské aplikace pro jakékoliv použití.

Popis

Akce; umožňuje, aby Flash animace komunikovala s programem hostícím Flash Přehrávač. Ve Web browseru volá **fscommand** JavaScript funkci **moviename_Dofscommand** na HTML stránce obsahující Flash animaci, kde **moviename** je jméno Flash Přehrávače, které je určeno atributem **NAME** tagu **EMBED** nebo vlastností **ID** tagu **OBJECT**. Jestliže je Flash Přehrávač určen jménem **Movie**, je volaná JavaScript funkce **Movie_Dofscommand**.

Přehrávač

Flash 3 nebo novější.

function

Syntaxe

```
function funkcejméno ([argument0,argument1,...argumentN]){
příkaz(y)
}
function ([argument0,argument1,...argumentN]){
příkaz(y)
}
```

Argumenty

funkcejméno Jméno nové funkce.

argument Nula nebo více řetězců, čísel nebo objektů, které se mají vložit do funkce.

příkazy Nula nebo více ActionScript zpráva, které jste definovali pro tělo funkce.

Popis

Akce; sada příkazů definovaných pro provedení určitých úkolů. Můžete **deklarovat** nebo definovat funkci v jedné lokalitě a volat nebo vyvolat ji z různých skriptů v animaci. Když definujete funkci, můžete pro ni určit také argumenty. Argumenty jsou schránky pro hodnoty, na kterých bude operovat funkce. Můžete vložit funkci s různými argumenty, také zvanými parametry, při každém jejím volání.

Aby funkce obdržela nebo generovala hodnotu, použijte ve funkcích **příkaz(y)** akci **return**.

Použití 1: Deklaruje **function** specifikovaným **funkcejméno, argumenty a příkaz(y)**. Když je funkce volána, je vyvolána deklarace funkce. Odkazování vpřed je povoleno; uvnitř stejného seznamu Akcí může být funkce deklarována poté, co je volána. Deklarace funkce nahrazuje jakékoliv předchozí deklarace stejné funkce. Můžete použít tuto syntaxi, kdekoliv je povolen příkaz.

Použití 2: Vytváří anonymní funkci a vrací ji. Tato syntaxe je použita ve výrazech a je zvláště užitečná pro instalování metod v objektech.

Přehrávač

Flash 5 nebo novější.

Příklad

(Použití 1) Následující příklad definuje funkci **sqr**, která akceptuje jeden argument a dává **square(x*x)** argumentu. Všimněte si, že jestliže je funkce deklarována a použita ve stejném skriptu, deklarace funkce se může objevit po použití funkce.

```
y = sqr(3);
function sqr(x){
return x*x;
}
```

(Použití 2) Následující funkce definuje objekt **Circle**:

```
function Circle(radius){
this.radius = radius;
}
```

Následující příkaz definuje anonymní funkci, která počítá rozlohu kruhu a připojuje ho k objektu **Circle** jako metodu:

```
Circle.prototype.rozloha = function()
{ return Math.PI*this.radius *this.radius }
```

ge (greather than or equal to – string specific)

Syntaxe
výraz1 `ge` **výraz2**

Argumenty
výraz1,výraz2 Čísla, řetězce nebo proměnné.

Popis
Operátor (srovnání); porovnává **výraz1** s **výrazem2** a udává **`true`**, jestliže je **výraz1** větší než nebo roven **výrazu2**; jinak udává **`false`**.

Přehrávač
Flash 4 nebo novější. Tento operátor byl zavržen ve Flash 5; je doporučováno používat nový operátor >=.

Viz také
`>=  (greater than or equal to)`

getProperty

Syntaxe
```
getProperty(instancejméno,vlastnost);
```

Argumenty
instancejméno Jméno instance movie klipu, pro který je vlastnost získávána.

vlastnost Vlastnost movie klipu, jako ja x nebo y osa.

Popis
Funkce; udává hodnotu specifikované vlastnosti pro movie klip instanci.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující příklad získává horizontální osu (**`_x`**) pro movie klip mojeMovie:
```
getProperty(_root.mojeMovie_polozka._x);
```


getTimer

Syntaxe
```
getTimer();
```

Argumenty
Žádné.

Popis
Funkce; udává počet milisekund, které uplynuly od doby, kdy začalo movie hrát.

Přehrávač
Flash 4 nebo novější.

getURL

Syntaxe
```
getURL(url [, okno [, proměnné ]]);
```

Argumenty
url URL, ze kterého se má získat dokument. URL musí být ve stejné subdoméně, jako URL, kde je movie aktuálně umístěno.

window
(okno) Volitelný argument specifikující okno nebo HTML rám, na který by měl být dokument natažen. Vložte jméno určitého okna nebo zvolte z následujících rezervovaných jmen cílů:

`_self` určuje aktuální rám v aktuálním okně.
`_blank` určuje nové okno.
`_parent` určuje rodiče aktuálního rámu.
`_top` určuje nejvyšší rám v aktuálním okně.

proměnné Volitelný argument určující metodu posílání proměnných. Jestliže zde nejsou žádné proměnné, tento argument vynechte; jinak specifikujte, zda natáhnout proměnné použitím metody **`GET`** nebo **`POST`**. **`GET`** připojí proměnné na konec **`URL`** a je použit pro malá množství proměnných. POST posílá proměnné v odděleném HTTP a je použit pro dlouhé řetězce proměnných.

Popis
Akce; natahuje dokument z určitého URL do okna nebo vkládá proměnné do další aplikace na definovaném URL. Pro testování této akce se ujistěte, že soubory, které mají být nataženy, jsou v určeném umístění. Pro použití absolutního URL (například **http://www.mujserver.com**) potřebujete síťové spojení.

Přehrávač
Flash 2 nebo novější. Volby GET a POST jsou dostupné pouze pro Flash 4 a pozdější verze Přehrávače.

Příklad
Tento příklad natahuje nové URL do prázdného okna browseru. Akce **getURL** cíluje proměnnou **prijemAd** jako parametr **url**, takže můžete změnit nataženou URL bez editace Flash movie. Hodnota proměnné **prijemAd** je propuštěna do Flash dříve v movie používající akci **loadVariables**.

```
on(release){
        getURL(prijemAd, "_blank");
}
```

Viz také
loadVariables
XML.send
XML.sendAndLoad
XMLSocket.send

getVersion

Syntaxe
```
getVersion();
```

Argumenty
Žádné.

Popis
Funkce; udává řetězec obsahující verzi Flash Přehrávače a informaci o platformě. Tato funkce nefunguje v režimu test-movie a udává informaci pouze pro verzi 5 nebo dále Flash Přehrávače.

Příklad
Následující je příkladem řetězce získaného funkcí getVersion:
```
WIN 5,0,17,0
```

Toto indikuje, že platformou je Windows a číslo verze Flash Přehrávače je hlavní verze 5, menší verze 17(5.0r17).

Přehrávač
Flash 5 nebo novější.

gotoAndPlay

Syntaxe
```
gotoAndPlay(scéna,snímek);
```

Argumenty
scéna	Jméno scény, na kterou je poslána hrací hlava.
snímek	Číslo snímku, na který je poslána hrací hlava.

Popis
Akce; posílá hrací hlavu na určitý snímek ve scéně a hraje od tohoto snímku. Jestliže není specifikována žádná scéna, hrací hlava jde na určitý snímek v aktuální scéně.

Přehrávač
Flash 2 nebo novější.

Příklad
Když uživatel klikne na tlačítko, ke kterému je připojena funkce **gotoAndPlay**, je hrací hlava poslána na snímek 16 a začne hrát.

```
on(release){
        gotoAndPlay(16);
}
```

gotoAndStop

Syntaxe
`gotoAndStop(scéna,snímek);`

Argumenty
scéna Jméno scény, na kterou je poslána hrací hlava.

snímek Číslo snímku, na který je poslána hrací hlava.

Popis
Akce; posílá hrací hlavu na specifikovaný snímek na scéně a zastavuje ji. Jestliže není specifikována žádná scéně, je hrací hlava poslána na snímek v aktuální scéně.

Přehrávač
Flash 2 nebo novější.

Příklad
Když uživatel klikne na tlačítko, ke kterému je připojena akce **gotoAndStop**, hrací hlava je poslána na snímek 5 a animace se zastaví.

```
on(releaset){
       gotoAndStop (5);
}
```

gt (greater than – string specific)

Syntaxe
výraz1 gt výraz2

Argumenty
výraz1,výraz2 Čísla, řetězce nebo proměnné.

Popis
Operátor (srovnání); porovnává výraz1 s **výrazem2** a udává true, jestliže je **výraz1** větší než **výraz2**; jinak udává false.

Přehrávač
Flash 4 nebo novější. Tento operátor byl ve Flash 5 zavržen; je doporučováno používat nový operátor >.

Viz také
`> (greater than)`

_height

Syntaxe
```
instancejméno._height
instancejméno._height = hodnota;
```

Argumenty
instancejméno Jméno instance movie klipu, pro který má být nastavena nebo získána vlastnost **_height.**

hodnota Celé číslo určující výšku movie klipu v pixelech.

Popis
Vlastnost; nastavuje a získává výšku prostoru okupovaného obsahem movie. V předchozích verzích Flash, byly vlastnosti **_height** a **_width** pouze pro čtení; ve Flash 5 mohou být tyto vlastnosti nastaveny.

Přehrávač
Flash 4 nebo novější.

Příklad
Následující kód nastavuje výšku a šířku movie klipu, když uživatel klikne na myš:

```
onClipEvent(mouseDown){
       _width=200;
       _height=200;
}
```

_highquality

Syntaxe

```
_highquality = hodnota ;
```

Argumenty

hodnota Úrovně anti-aliasingu (vyrovnání) aplikovaná na animaci. Specifikuj 2 (Best) pro aplikaci vysoké kvality se vždy hladkými bitmapami. Specifikuj 1 (High quality) pro aplikaci anti-aliasing; toto uhladí bitmapy, pokud animace neobsahuje jejich animaci. Specifikuj 0 (low quality) pro zabránění anti-aliasingu.

Popis

Vlastnost (všeobecná); specifikuje úroveň anti-aliasingu aplikovanou na aktuální animaci.

Přehrávač

Flash 4 nebo novější.

Viz také

`_quality`
`toggleHighQuality`

if

Syntaxe

```
if(podmínka){
příkaz;
}
```

Argumenty

podmínka Výraz, který se ohodnocuje na `true` nebo `false`. Například `if(name == „Erica")` ohodnotí proměnnou `name`, zda je „Erica".

příkaz Instrukce pro vykonání `if`, nebo když se podmínka ohodnotí na `true`.

Popis

Akce; ohodnocuje podmínku pro určení další akce v animaci. Jestliže je podmínka `true`, Flash vykonává příkazy, které následují. Použijte `if` pro vytvoření logických větví ve vašich skriptech.

Přehrávač

Flash 4 nebo novější.

Viz také

`else`
`for`
`for…in`

ifFrameLoaded

Syntaxe

```
ifFrameLoaded(scéna,snímek){
příkaz;
}

ifFrameLoaded (snímek){
příkaz;
}
```

Argumenty

scéna Scéna, na kterou se ptáme.

snímek Číslo snímku nebo popis snímku, který se má natáhnout před vykonáním dalšího příkazu.

Popis

Akce; kontroluje, zda jsou místně dostupné obsahy určitého snímku. Použijte `ifFrameLoaded` pro začátek hraní jednoduché animace, zatímco se do místního počítače natahuje zbytek movie. Rozdíl mezi použitím `_framesloaded` a `ifFrameLoaded` je ten, že `_framesloaded` vám umožňuje přidat příkaz `if` nebo `else`, zatímco akce `ifFrameLoaded` vám umožňuje určit počet snímků v jednom jednoduchém příkazu.

Přehrávač

Flash 3 nebo novější. Akce `ifFrameLoaded` je ve Flash 5 zavržena; je doporučováno používat akci `_framesloaded`.

Viz také

`_framesloaded`

#include

Syntaxe
```
#include „souborjméno.as“;
```

Argumenty
souborjméno.as Jméno souboru, který má být zahrnut; as je doporučovaná přípona souboru.

Popis
Akce; zahrnuje obsah souboru určeného v argumentu, když je animace testována, publikována nebo exportována. Akce **`#include`** je volána, když testujete, publikujete nebo exportujete. Akce **`#include`** je kontrolována, když se objeví kontrola syntaxe.

Přehrávač
N/A

Infinity

Syntaxe
```
Infinity
```

Argumenty
Žádné.

Popis
Nejvyšší proměnná; předdefinovaná proměnná s ECMA-262 hodnotou pro nekonečnost.

Přehrávač
Flash 5 nebo novější.

int

Syntaxe
```
int(hodnota);
```

Argumenty
hodnota Číslo, které má být zaokrouhleno na celé číslo.

Popis
Funkce; konvertuje decimální čísla na nejbližší hodnotu celého čísla.

Přehrávač
Flash 4 nebo novější. Tato funkce byla ve Flash 5 zavržena; je doporučováno používat metodu **`Math.floor`**.

Viz také
`Math.floor`

isFinite

Syntaxe
```
isFinite(výraz);
```

Argumenty
výraz Booleovská, proměnná nebo jiný výraz, který má být ohodnocen.

Popis
Nejvyšší funkce; ohodnocuje argument a udává **`true`**, jestliže je číslo konečné, a **`false`**, jestliže je nekonečné nebo negativně nekonečné. Přítomnost nekonečného nebo negativně nekonečného indikuje podmínka matematické chyby, jako je dělení nulou.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující jsou příklady obdržení hodnot pro **`isFinite`**:
`isFinite (56)` dává **`true`**

`isFinite (Number.POSITIVE_INFINITY)` udá **`false`**

`isNaN (Number.POSITIVE_INFINITY)` udá **`false`**

isNaN

Syntaxe

```
isNaN(výraz);
```

Argumenty

výraz Booleovská, proměnná nebo jiný výraz, který má být ohodnocen.

Popis

Nejvyšší funkce; ohodnocuje argument a udává **true**, jestliže hodnota není číslo (**NaN**), indikuje přítomnost matematických chyb.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující ilustruje získanou hodnotu pro **isNaN**:

isNaN(„Strom“) udá **true**

isNaN(56) udá **false**

isNaN(Number.POSITIVE_INFINITY) udá **false**

Key (object)

Objekt Key (Klávesa) je objekt nejvyšší úrovně, který můžete dosáhnout bez použití konstruktoru. Pro vytvoření rozhraní, které může být ovládáno uživatelem standardní klávesnicí, použijte metody objektu Key. Vlastnosti objektu Key jsou konstanty reprezentující klávesy, které jsou nejběžněji používány například pro řízení her. Viz Dodatek B, „Klávesy Klávesnice a Hodnoty Kódů Kláves“, kde je kompletní seznam hodnot klávesových kódů.

Příklad

```
onClipEvent (enterFrame){
        if(Key.isDown(Key.RIGHT)){
                setProperty ("", _x, _x+10);
        }
}
```

nebo

```
onClipEvent (enterFrame){
        if(Key.isDown(39)){
                setProperty("", _x, _x+10);
        }
}
```

Přehled metod objektu Key

Metoda	Popis
`getAscii;`	Udává ASCII hodnotu naposledy stisknuté klávesy.
`getCode;`	Udává virtuální klávesový kód naposledy stisknuté klávesy.
`isDown;`	Udává **true**, jestliže je klávesa určená v argumentu stisknuta.
`isToggled;`	Udává **true**, jestliže je aktivována klávesa Num Lock nebo Caps Lock.

Přehled vlastností objektu Key

Všechny vlastnosti objektu Key jsou konstanty.

Metoda	Popis
`BACKSPACE`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Backspace (9).
`CAPSLOCK`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Caps Lock (20).
`CONTROL`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Control (17).
`DELETEKEY`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Delete (46).
`DOWN`	Konstanta spojená s hodnotou kódu klávesy pro klávesu ŠipkaDolů (40).
`END`	Konstanta spojená s hodnotou kódu klávesy pro klávesu End (35).
`ENTER`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Enter (13).
`ESCAPE`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Escape (27).
`HOME`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Home (36).
`INSERT`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Insert (45).
`LEFT`	Konstanta spojená s hodnotou kódu klávesy pro klávesu ŠipkaVlevo (37).
`PGDN`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Page Down (34).
`PGUP`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Page Up (33).
`RIGHT`	Konstanta spojená s hodnotou kódu klávesy pro klávesu ŠipkaVpravo (39).
`SHIFT`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Shift (16).
`SPACE`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Space (32).
`TAB`	Konstanta spojená s hodnotou kódu klávesy pro klávesu Tab (9).
`UP`	Konstanta spojená s hodnotou kódu klávesy pro klávesu ŠipkaNahoru (38).

Key.BACKSPACE

Syntaxe

```
Key.BACKSPACE
```

Argumenty

Žádné.

Popis

Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Backspace (9).

Přehrávač

Flash 5 nebo novější.

Key.CAPSLOCK

Syntaxe

```
Key.CAPSLOCK
```

Argumenty

Žádné.

Popis

Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Caps Lock (20).

Přehrávač

Flash 5 nebo novější.

Key.CONTROL

Syntaxe

```
Key.CONTROL
```

Argumenty

Žádné.

Popis

Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Control (17).

Přehrávač

Flash 5 nebo novější.

Key. DELETEKEY

Syntaxe

```
Key.DELETEKEY
```

Argumenty

Žádné.

Popis

Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Delete (46).

Přehrávač

Flash 5 nebo novější.

Key.DOWN

- **Syntaxe**
  `Key.DOWN`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Šipka Dolů (40).

- **Přehrávač**
  Flash 5 nebo novější.

Key.END

- **Syntaxe**
  `Key.END`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu End (35).

- **Přehrávač**
  Flash 5 nebo novější.

Key.ENTER

- **Syntaxe**
  `Key.ENTER`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou klávesy pro klávesu Enter (13).

- **Přehrávač**
  Flash 5 nebo novější.

Key.ESCAPE

- **Syntaxe**
  `Key.ESCAPE`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Escape (27).

- **Přehrávač**
  Flash 5 nebo novější.

Key.getAscii

- **Syntaxe**
  `Key.getAscii();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává ASCII kód naposledy stisknuté nebo uvolněné klávesy.

- **Přehrávač**
  Flash 5 nebo novější.

Key.getCode

- **Syntaxe**
  `Key.getCode();`

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává hodnotu klávesového kódu naposledy stisknuté klávesy. Použijte informaci v Dodatku B, „Klávesy Klávesnice a Hodnoty Kódů Kláves“, pro porovnání udaného kódu klávesy s virtuální klávesou na standardní klávesnici.

- **Přehrávač**
  Flash 5 nebo novější.

Key.HOME

- **Syntaxe**
  `Key.HOME`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Home (36).

- **Přehrávač**
  Flash 5 nebo novější.

Key.INSERT

- **Syntaxe**
  `Key.INSERT`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Insert (45).

- **Přehrávač**
  Flash 5 nebo novější.

Key.isDown

- **Syntaxe**
  `Key.isDown(kódklávesy);`

- **Argumenty**
  kódklávesy Hodnota kódu klávesy spojená s určitou klávesou, nebo vlastnost objektu Key spojená s určitou klávesou. V Dodatku B, „Klávesy Klávesnice a Hodnoty Kódů Kláves“, jsou zapsány všechny kódy kláves spojené s klávesami na standardní klávesnici.

- **Popis**
  Metoda; udává **`true`**, jestliže je klávesa určená v **kódklávesy** stisknutá. Na Macintosh jsou hodnoty kódů kláves pro Caps Lock a Num Lock identické.

- **Přehrávač**
  Flash 5 nebo novější.

Key.isToggled

- **Syntaxe**
  `Key.isToggled`**(kódklávesy)**

- **Argumenty**
  kódklávesy Kód klávesy pro Caps Lock (20) nebo Num Lock (144).

- **Popis**
  Metoda; udává **`true`**, jestliže je aktivována (spuštěna) klávesa Caps Lock nebo Num Lock.
  Na Macintosh jsou hodnoty kódů kláves pro tyto klávesy identické.

- **Přehrávač**
  Flash 5 nebo novější.

Key.LEFT

- **Syntaxe**
  `Key.LEFT`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Šipka Vlevo (37).

- **Přehrávač**
  Flash 5 nebo novější.

Key.PGDN

- **Syntaxe**
  `Key.PGDN`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Page Down (34).

- **Přehrávač**
  Flash 5 nebo novější.

Key.PGUP

- **Syntaxe**
  `Key.PGUP`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Page Up (33).

- **Přehrávač**
  Flash 5 nebo novější.

Key.RIGHT

- **Syntaxe**
  `Key.RIGHT`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Šipka Vpravo (39).

- **Přehrávač**
  Flash 5 nebo novější.

Key.SHIFT

- **Syntaxe**
  `Key.SHIFT`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Shift (16).

- **Přehrávač**
  Flash 5 nebo novější.

Key.SPACE

- **Syntaxe**
  `Key.SPACE`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Space (32).

- **Přehrávač**
  Flash 5 nebo novější.

Key.TAB

- **Syntaxe**
  `Key.TAB`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Tab (9).

- **Přehrávač**
  Flash 5 nebo novější.

Key.UP

- **Syntaxe**
  `Key.UP`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; konstanta spojená s hodnotou kódu klávesy pro klávesu Šipka Nahoru (38).

- **Přehrávač**
  Flash 5 nebo novější.

le (less than or equal to – string specific)

- **Syntaxe**
  výraz1 `le` **výraz2**

- **Argumenty**
  výraz1,**výraz2** Čísla, řetězce nebo proměnné.

- **Popis**
  Operátor (srovnání); porovnává **výraz1** a **výraz2** a udává `true`, jestliže je **výraz1** menší než nebo roven **výrazu2**; jinak udává `false`.

- **Přehrávač**
  Flash 4 nebo novější. Tento operátor byl ve Flash 5 zavržen; je doporučeno používat nový operátor **<=**.

- **Viz také**
  `<= (less than or equal to)`

length

- **Syntaxe**
  `length`**(výraz);**
  `length`**(proměnná);**

- **Argumenty**
  výraz Jakýkoliv řetězec.

  proměnná Jméno proměnné.

- **Popis**
  Řetězcová funkce; udává délku určeného řetězce nebo jméno proměnné.

- **Přehrávač**
  Flash 4 nebo novější. Tato funkce, spolu se všemi řetězcovými funkcemi, byla ve Flash 5 zavržena. Je doporučeno používat metody a vlastnost `length` objektu `String` (Řetězec), pro provedení stejných operací.

- **Příklad**
  Následující příklad udává hodnotu řetězce Ahoj:
  `length(„Ahoj“);`

  Výsledek je `4`.

- **Viz také**
  `“” (String delimiter)`
  `String.length`

_level

Syntaxe

```
_level N;
```

Argumenty

N Nezáporné celé číslo určující úroveň hloubky. V základním nastavení je **_level** nastavena na 0, movie je v základu hierarchie.

Popis

Vlastnost; odkaz na kořenovou Časovou osu animace **levelN**. Animaci musíte natáhnout pomocí akce **loadMovie** dříve, než ji cílujete použitím vlastnosti **_level**.

Ve Flash Přehrávači jsou animace určeny čísly podle pořadí, v jakém byly nataženy. Animace, která byla natažena první, je natažena na nejnižší úroveň, úroveň 0. Animace v úrovni 0 nastavuje míru snímku, barvu pozadí a velikost snímku pro všechny následně natažené animace. Ty jsou potom vrženy do vyšších úrovní nad animaci v úrovni 0. Úroveň, ve které je animace umístěna, je nazývána také úroveň hloubky nebo hloubka (depth).

Přehrávač

Flash 4 nebo novější.

Příklad

Následující příklad zastavuje Časovou osu animace v úrovni 0:

```
_level0.stop();
```

Následující příklad posílá Časovou osu animace v úrovni 4 na snímek 5. Animace v úrovni 4 musí být předtím natažena akcí loadMovie:

```
_level4.gotoAndStop(5);
```

Viz také

loadMovie
MovieClip.swapDepths

loadMovie

Syntaxe

```
loadMovie(url [, umístění/cíl, proměnné]);
```

Argumenty

url Absolutní nebo relativní URL pro SWF soubor, který se má natáhnout. Relativní cesta musí být relativní k SWF. URL musí být ve stejné subdoméně jako URL, kde animace aktuálně umístěna. Pro použití ve Flash Přehrávači, nebo pro testování v režimu test-movie v autorském prostředí Flash, musí být všechny SWF soubory uloženy ve stejné složce a jména souborů nemohou zahrnovat specifikace složky nebo disku.

cíl Volitelný argument určující cílový movie klip, který bude nahrazen nataženou animací. Natažená animace zdědí vlastnosti pozice, rotace a velikost cílového movie klipu. Určení do cíle je stejné jako určení do umístění (úroveň); neměli byste používat oba způsoby.

umístění Volitelný argument určující úroveň, do které je animace natažena. Natažené animace zdědí vlastnosti pozice, rotace a velikost cílového movie klipu. Pro natažení nové animace k již existujícím animacím určete úroveň, která není obsazena jinou animací. Pro nahrazení existující animace nataženou animace určete úroveň, která je aktuálně obsazena jinou animací. Pro nahrazení původního animace a stažení všech úrovní, natáhněte novou animaci do úrovně 0. Animace v úrovni 0 nastavuje měřítko rámečku, barvu pozadí a velikost rámečku pro všechny další natažené animace.

proměnné Volitelný argument určující způsob pro posílání proměnných spojených s animací, která má být natažena. Argument musí být řetězec „GET“ nebo „POST“. Jestliže zde nejsou žádné proměnné, tento argument vynechte; jinak určete, zda natáhnout proměnné použitím způsobu GET nebo POST. GET přiřadí proměnné na konec URL a je používán pro malý počet proměnných. POST pošle proměnné v odděleném HTTP a je používán pro dlouhé řetězce proměnných.

Popis

Akce; přehrává externí animace bez zavírání Flash Přehrávače. Normálně Flash Přehrávač zobrazí jednu animaci (SWF soubor) a potom se zavře. Akce **loadMovie** vás nechá zobrazit několik animací najednou, nebo mezi nimi přepínat bez natahování dalšího HTML dokumentu.

Můžete také natáhnout animace do úrovně, která již má natažené SWF soubory. Jestliže to uděláte, nová animace nahradí existující SWF soubor. Jestliže natáhnete novou animace do Úrovně 0, každá úroveň je stažena a Úroveň 0 je nahrazena novým souborem. Použijte akci **loadVariables** pro udržení aktivity animace a aktualizujte proměnné novými hodnotami.
Pro odstranění animací natažených akcí **loadMovie** použijte akci unloadMovie.

Přehrávač

Flash 3 nebo novější.

Příklad

Tento příkaz **loadMovie**, je připojen k navigačnímu tlačítku označenému Produkty. Na Scéně je neviditelný movie klip s instancí jménem **pustitZona**. Akce **loadMovie** používá tento movie klip jako cílový parametr pro natažení produktů v SWF souboru do správného umístění na Scéně:

```
on(release){
        loadMovie(„produkty.swf", _root.pustitZona);
}
```

Viz také

unloadMovie
_level

loadVariables

Syntaxe

loadVariables(url, umístění [, proměnné]);

Argumenty

url — Absolutní nebo relativní URL, kde jsou umístěny proměnné. Hostitelem pro URL musí být stejná subdoména jako pro animaci.

umístění — Úroveň nebo cíl pro získání proměnných. Ve Flash Přehrávači jsou animace označeny číslem, které se vztahuje k pořadí v jakém byly nataženy. První animace se natahuje do spodní úrovně **(level 0)**. Uvnitř akce **loadMovie** musíte určit číslo úrovně pro každou následnou animaci. Tento argument je volitelný.

proměnné — Volitelný argument určující způsob pro posílání proměnných. Jestliže zde nejsou žádné proměnné, tento argument vynechte; jinak určete, zda natáhnout proměnné pomocí použití způsobu GET nebo POST. GET připojí proměnné na konec URL a je používán pro malé množství proměnných. POST posílá proměnné v odděleném HTTP a je používán pro dlouhé řetězce proměnných.

Popis

Akce; čte data z externího souboru, jako je textový soubor nebo text generovaný CGI skriptem, Active Server Pages (ASP) nebo Personal Home Page (PHP) a nastavuje hodnoty proměnných v animaci nebo movie klipu. Tato akce může být také použita pro updatování proměnných v aktivní animaci novými hodnotami.

Text ve specifikované URL musí být ve standardním MIME formátu **application/x-www-urlformencoded** (standardní formát používaný CGI skripty). Animace a proměnné, které mají být nataženy musí sídlit ve stejné subdoméně. Může být určen jakýkoliv počet proměnných. Například věta dole definuje několik proměnných:

```
spolecnost=Macromedia&adresa=600+Mestoposlat&mesto=
San+Francisco&zip=94
103
```

Přehrávač

Flash 4 nebo novější.

- **Příklad**
  Tento příklad natahuje informaci z textového souboru do textového pole v hlavní Časové ose (level 0). Jména proměnných textového pole musí souhlasit se jmény proměnných v souboru data.txt.

```
on(release){
        loadVariables(„data.txt", 0);
}
```

- **Viz také**
  `getURL`
  `MovieClip.loadMovie`
  `MovieClip.loadVariables`

lt (less than – string specific)

- **Syntaxe**
  výraz1 `lt` **výraz2**

- **Argumenty**
  výraz1,výraz2 Čísla, řetězce nebo proměnné.

- **Popis**
  Operátor (srovnání); porovnává **výraz1** s **výrazem2** a udává `true`, jestliže je **výraz1** menší než **výraz2**; jinak udává false.

- **Přehrávač**
  Flash 4 nebo novější. Tento operátor byl ve Flash 5 zavržen; doporučuje se používat nový operátor `< (less than)`.

- **Viz také**
  `< (less than)`

Math (object)

Objekt Math je objekt nejvyšší úrovně, kterého můžete dosáhnout bez použití konstruktoru.

Metody a vlastnosti tohoto objektu použijte pro dosažení a manipulaci s matematickými konstantami a funkcemi. Všechny vlastnosti a metody objektu Math jsou statické a musí být volány použitím syntaxe `Math.metoda(argument)` nebo `Math.konstanta`. V ActionScriptu jsou konstanty definovány s maximální precizností dvojité přesnosti IEEE-754 plynoucích čísel.

Objekt Math je plně podporován ve Flash 5 Přehrávači. Ve Flash 4 Přehrávači, metody objektu Math fungují, ale jsou napodobeny použitím odhadů a nemohou být přesné jako nenapodobené funkce podporované Flash 5 Přehrávačem.

Některé metody objektu `Math` mají argument radián úhlu. Můžete použít citaci dole pro kalkulaci hodnot radiánů nebo jednoduše zadat rovnici (vložení hodnoty pro stupně) pro argument radián.

Pro kalkulaci hodnoty radiánu použijte tento vzorec:
`radian=Math.PI/180*stupně`

Následující je příkladem propuštění vzorce jako argumentu pro výpočet sinu úhlu 45 stupňů:
`Math.SIN(Math.PI/180*45) je stejné jako Math.SIN(.7854)`

Přehled metod objektu Math

Metoda	Popis
`abs`	Vypočítá absolutní hodnotu.
`acos`	Vypočítá arc cosinus.
`asin`	Vypočítá arc sinus.
`atan`	Vypočítá arc tangent.
`atan2`	Vypočítá úhel z osy x do bodu.
`ceil`	Zaokrouhlí číslo nahoru na nejbližší celé číslo.
`cos`	Vypočítá cosinus.
`exp`	Vypočítá exponenciální hodnotu.
`floor`	Zaokrouhlí číslo dolů na nejbližší celé číslo.
`log`	Vypočítá přirozený logaritmus.
`max`	Udává větší ze dvou celých čísel.
`min`	Udává menší ze dvou celých čísel.
`pow`	Vypočítá x zvýšené na mocninu y.
`random`	Udává pseudo-náhodné číslo mezi 0.0 a 1.0.
`round`	Zaokrouhluje na nejbližší celé číslo.
`sin`	Vypočítá sinus.
`sqrt`	Vypočítá čtvercový kořen (odmocninu).
`tan`	Vypočítá tangent.

Přehled vlastností pro objekt Math

Všechny vlastnosti pro objekt Math jsou konstanty.

Metoda	Popis
`E`	Eulerova konstanta a základ přirozeného logaritmu (přibližně 2,718).
`LN2`	Přirozený logaritmus dvou (přibližně 0,693).
`LOG2E`	Základ 2 logaritmu e (přibližně 1,442).
`LN10`	Přirozený logaritmus 10 (přibližně 2,302).
`LOG10E`	Základ 10 logaritmu e (přibližně 0,434).
`PI`	Poměr obvodu kruhu k jeho průměru (přibližně 3,14159).
`SQRT1_2`	Reciproční kořenu čtverce (odmocnina) 1/2 (přibližně 0,707).
`SQRT2`	Kořen čtverce (odmocnina) 2 (přibližně 1,414).

Math.abs

- **Syntaxe**

  Math.`abs(x);`

- **Argumenty**

  x Jakékoliv číslo.

- **Popis**

  Metoda; vypočítává a udává absolutní hodnotu čísla určeného argumentem **x**.

- **Přehrávač**

  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.acos

- **Syntaxe**

  Math.`acos(x);`

- **Argumenty**

  x Číslo od -1,0 do 1,0.

- **Popis**

  Metoda; vypočítává a udává arc cosinus čísla určeného v argumentu **x**, v radiánech.

- **Přehrávač**

  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.asin

- **Syntaxe**

  Math.`asin(x);`

- **Argumenty**

  x Číslo od -1,0 do 1,0.

- **Popis**

  Metoda; vypočítává a udává arc sinus pro číslo určené v argumentu **x**, v radiánech.

- **Přehrávač**

  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.atan

Syntaxe

Math.`atan(x);`

Argumenty

x Jakékoliv číslo.

Popis

Metoda; vypočítává a udává arc tangent pro číslo určené v argumentu **x**. Obdržená hodnota je mezi záporným pí děleným 2, a kladným pí děleným 2.

Přehrávač

Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.atan2

Syntaxe

Math.`atan2(y, x);`

Argumenty

x Číslo určující osu **x** bodu.

y Číslo určující osu **y** bodu.

Popis

Metoda; vypočítává a udává arc tangent **y**/**x** v radiánech. Obdržená hodnota reprezentuje úhel naproti protějšímu úhlu pravého trojúhelníku, kde **x** je délka přilehlé strany a **y** je délka protilehlé strany.

Přehrávač

Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.ceil

Syntaxe

Math.`ceil(x);`

Argumenty

x Číslo nebo výraz.

Popis

Metoda; udává ceiling (horní limit) určitého čísla nebo výrazu. Limitem pro číslo je nejbližší celé číslo, které je větší nebo rovno číslu.

Přehrávač

Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.cos

Syntaxe

Math.`cos(x);`

Argumenty

Úhel měřený v radiánech.

Popis

Metoda; udává cosinus (hodnota od -1,0 do 1,0) úhlu určeného argumentem **x**. Úhel **x** musí být určený v radiánech. Pro výpočet radiánu použijte informaci v úvodu do objektu **Math**.

Přehrávač

Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.E

Syntaxe
Math.E

Argumenty
Žádné.

Popis
Konstanta; matematická konstanta pro základ přirozených logaritmů, vyjádřená jako **e**. Přibližná hodnota **e** je **2,71828.**

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.exp

Syntaxe
Math.exp(x);

Argumenty
x Exponent; číslo nebo výraz.

Popis
Metoda; udává hodnotu základu přirozeného logaritmu **(e)** umocněného exponentem určeným argumentem **x**. Konstanta Math.E může poskytnout hodnotu **e**.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.floor

Syntaxe
Math.floor(x);

Argumenty
x Číslo nebo výraz.

Popis
Metoda; udává dno čísla nebo výrazu určeného v argumentu **x**. Dno je nejbližší celé číslo menší nebo rovno určenému číslu nebo výrazu.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Příklad
Následující udává hodnotu 12:

```
Math.floor(12,5);
```

Math.log

Syntaxe
Math.log(x);

Argumenty
x Číslo nebo výraz s hodnotou vyšší než 0.

Popis
Metoda; udává přirozený logaritmus argumentu **x**.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.LOG2E

- **Syntaxe**
  Math.LOG2E

- **Argumenty**
  Žádné.

- **Popis**
  Konstanta; matematická konstanta pro základ-2 logaritmu konstanty **e (Math.E)**, vyjádřená jako **log e 2**, s přibližnou hodnotou 1,442695040888963387.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.LOG10E

- **Syntaxe**
  Math.LOG10E

- **Argumenty**
  Žádné.

- **Popis**
  Konstanta; matematická konstanta pro základ-10 logaritmu konstanty **e (Math.E)**, vyjádřená jako **log 10 e**, s přibližnou hodnotou 0,4342944819032518167.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.LN2

- **Syntaxe**
  Math.LN2

- **Argumenty**
  Žádné.

- **Popis**
  Konstanta; matematická konstanta pro přirozený logaritmus 2, vyjádřený jako **log e 2**, s přibližnou hodnotou 0,6931471805599452862.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.LN10

- **Syntaxe**
  Math.LN10

- **Argumenty**
  Žádné.

- **Popis**
  Konstanta; matematická konstanta pro přirozený logaritmus 10, vyjádřená jako **log e 10**, s přibližnou hodnotou 2,302585092994045901.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.max

- **Syntaxe**
  Math.max(x, y);

- **Argumenty**
  x Číslo nebo výraz.

  y Číslo nebo výraz.

- **Popis**
  Metoda; ohodnocuje **x** a **y** a udává větší hodnotu.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.min

- **Syntaxe**
  Math.min (x, y);

- **Argumenty**
  x Číslo nebo výraz.

  y Číslo nebo výraz.

- **Popis**
  Metoda; ohodnocuje **x** a **y** a udává menší hodnotu.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.PI

- **Syntaxe**
  Math.PI

- **Argumenty**
  Žádné.

- **Popis**
  Konstanta; matematická konstanta pro poměr obvodu kruhu k jeho průměru, vyjádřená jako pí, s hodnotou 3,14159265358979.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.pow

- **Syntaxe**
  Math.pow (x, y);

- **Argumenty**
  x Číslo, které má být zvýšeno na mocninu.

  y Číslo určující mocninu, na kterou se má **x** zvýšit.

- **Popis**
  Metoda; vypočítává a udává **x** umocněné **y**, **xy**.

- **Přehrávač**
  Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.random

Syntaxe
Math.`random();`

Argumenty
Žádné.

Popis
Metoda; udává pseudo-náhodné číslo mezi 0,0 a 1,0.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Viz také
`random`

Math.round

Syntaxe
Math.`round(x);`

Argumenty
x Jakékoliv číslo.

Popis
Metoda; zaokrouhluje hodnotu argumentu **x** nahoru nebo dolů na nejbližší celé číslo a udává hodnotu.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.sin

Syntaxe
Math.`sin(x);`

Argumenty
x Úhel měřený v radiánech.

Popis
Metoda; vypočítává a udává sinus určeného úhlu v radiánech. Pro výpočet radiánu použijte informaci v úvodu do objektu **Math**.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické unkce podporované Flash 5 Přehrávačem.

Viz také
`Math (object)`

Math.sqrt

Syntaxe
Math.`sqrt(x);`

Argumenty
x Jakékoliv číslo nebo výraz větší než nebo roven 0.

Popis
Metoda; vypočítává a udává odmocninu určitého čísla.

Přehrávač
Flash 5 nebo novější. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusí být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.SQRT1_2

Syntaxe
Math.`SQRT1_2`

Argumenty
Žádné.

Popis
Konstanta; matematická konstanta pro recipročnı́ odmocniny jedné poloviny (1/2), s přibližnou hodnotou 0,707106781186.

Přehrávač
Flash 5 nebo novějšı́. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusı́ být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.SQRT2

Syntaxe
Math.`SQRT2`

Argumenty
Žádné.

Popis
Konstanta; matematická konstanta pro odmocninu 2, s přibližnou hodnotou 1,414213562373.

Přehrávač
Flash 5 nebo novějšı́. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusı́ být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

Math.tan

Syntaxe
Math.`tan(x);`

Argumenty
x Úhel měřený v radiánech.

Popis
Metoda; vypočı́tává a udává tangent určeného úhlu. Použijte informaci z úvodu do objektu **Math**, pro kalkulaci radiánu.

Přehrávač
Flash 5 nebo novějšı́. Ve Flash 4 Přehrávači jsou metody a vlastnosti objektu **Math** simulovány použitím odhadů a nemusı́ být přesné jako nesimulované matematické funkce podporované Flash 5 Přehrávačem.

maxscroll

Syntaxe
proměnná_jméno`.maxscroll = x`

Argumenty
proměnná_jméno Jméno proměnné spojené s textovým polem

x Číslo řádku, je maximálnı́ hodnota možná pro vlastnost **`scroll`**, založená na výšce textového pole. Toto je hodnota pouze pro čtenı́, nastavená Flashem.

Popis
Vlastnost; vlastnost pouze pro čtenı́, která funguje s vlastnostı́ **`scroll`**, pro kontrolu zobrazenı́ informace v textovém poli. Tato vlastnost může být získána, ale nemůže být modifikována.

Přehrávač
Flash 4 nebo novějšı́.

Viz také
`scroll`

mbchr

Syntaxe

```
mbchr(číslo);
```

Argumenty

číslo Číslo, které se má konvertovat na multibytový znak.

Popis

Řetězcová funkce; konvertuje číslo v ASCII kódu na multibytový znak.

Přehrávač

Flash 4 nebo novější. Tato funkce byla ve Flash 5 zavržena; je doporučeno používat metodu **`String.fromCharCode`**.

Viz také

`String.fromCharCode`

mblength

Syntaxe

```
mblength(řetězec);
```

Argumenty

řetězec Řetězec.

Popis

Řetězcová funkce; udává délku řetězce multibytového znaku.

Přehrávač

Flash 4 nebo novější. Funkce byla ve Flash 5 zavržena; je doporučeno použít objekt `String` a jeho metody.

mbord

Syntaxe

```
mbord(znak);
```

Argumenty

znak Znak, který se má konvertovat na multibytové číslo.

Popis

Řetězcová funkce; konvertuje specifikovaný znak na multibytové číslo.

Přehrávač

Flash 4 nebo novější. Tato funkce byla ve Flash 5 zavržena; je doporučeno používat metodu **`String.charCodeAt`**.

Viz také

`String.frmCharCode`

mbsubstring

Syntaxe

`mbsubstring`(hodnota, index, počet);

Argumenty

value (hodnota) Multibytový řetězec, ze kterého se má vytáhnout nový multibytový řetězec.

index Číslo prvního znaku, který se má vytáhnout.

count (počet) Počet znaků, které mají být zahrnuty ve vytaženém řetězci, bez znaku index.

Popis

Řetězcová funkce; vytahuje nový multibytový znakový řetězec z multibytového znakového řetězce.

Přehrávač

Flash 4 nebo novější. Tato funkce byla ve Flash 5 zavržena; je doporučeno použití metody **`String.substr`**.

Viz také

`String.substr`

Mouse (object)

Použijte metody objektu Mouse pro schovávání a ukázání kurzoru v animaci. Ukazatel myši je v nastavení viditelný, avšak můžete ho schovat a zobrazit speciální kurzor, který si vytvoříte použitím movie klipu.

Přehled metod objektu Mouse

Metoda	Popis
`hide`	Schová kurzor v movie.
`show`	Zobrazí kurzor v movie.

Mouse.hide

- **Syntaxe**

  Mouse.`hide();`

- **Argumenty**

  Žádné.

- **Popis**

  Metoda; schovává kurzor v animaci. Kurzor je v nastavení viditelný.

- **Přehrávač**

  Flash 5 nebo novější.

- **Příklad**

  Následující kód připojený k movie klipu na hlavní Časové ose schová standardní kurzor a nastaví pozici **x** a **y** movie klip instance `specialniKurzor` na pozice **x** a **y** v hlavní Časové ose:

```
onClipEvent(enterFrame){
        Mouse.hide();
        specialniKurzorMC_x = _root._xmouse;
        specialniKurzorMC_y = _root._ymouse;
}
```

- **Viz také**

  `_xmouse`
  `_ymouse`
  `Mouse.show`

Mouse.show

- **Syntaxe**

  Mouse.`show();`

- **Argumenty**

  Žádné.

- **Popis**

  Metoda; učiní kurzor v animaci viditelný. Kurzor je v nastavení viditelný.

- **Přehrávač**

  Flash 5 nebo novější.

- **Viz také**

  `_xmouse`
  `_ymouse`
  `Mouse.hide`

MovieClip (object)

Metody pro objekt MovieClip poskytují stejnou funkčnost jako standardní akce, které cílují movie klipy. Jsou zde ale ještě dodatečné metody poskytující funkčnost nedosažitelnou při použití standardních akcí zapsaných v kategorii Akce panelu Akcí. Abyste volali metody objektu MovieClip, nemusíte používat konstrukční metody; namísto toho odkážete na movie klip instance jménem, při použití následující syntaxe:

```
jakykolivMovieClip.play();
jakykolivMovieClip.gotoAndPlay(3);
```

Přehled metod objektu MovieClip

Metoda	Popis
`attachMovie`	Připojuje movie klip z knihovny.
`duplicateMovieClip`	Duplikuje specifikovaný movie klip.
`getBounds`	Udává minimální a maximální souřadnice movie klipu x a y ve specifikovaném souřadnicovém prostoru.
`getBytesLoaded`	Udává počet bytů natažených pro specifikovaný movie klip.
`getBytesTotal`	Udává velikost movie klipu v bytech.
`getURL`	Konvertuje bodový objekt ze souřadnic Scény na lokální souřadnice specifikovaného movie klipu.
`globalToLocal`	Posílá hrací hlavu na specifikovaný snímek v movie klipu a hraje animaci.
`gotoAndPlay`	Posílá hrací hlravu na specifikovaný snímek v movie klipu a zastavuje animaci
`gotoAndStop`	Udává `true`, jestliže hraniční box specifikovaného movie klipu protíná hraniční box cílového movie klipu.
`hitTest`	Natahuje externí animaci do movie klipu.

Metoda	Popis
`loadMovie`	Natahuje externí animaci do movie klipu.
`loadVariables`	Natahuje proměnné z URL nebo jiného umístění do movie klipu.
`localToGlobal`	Konvertuje objekt Point (Bod) z lokálních souřadnic movie klipu na globální souřadnice Scény.
`nextFrame`	Posílá hrací hlavu na další snímek movie klipu.
`play`	Hraje specifikovaný movie klip.
`prevFrame`	Posílá hrací hlavu na předchozí snímek movie klipu.
`removeMovieClip`	Odstraňuje movie klip z Časové osy, jestliže byl vytvořen akcí nebo metodou `duplicateMovieClip` nebo metodou `attachMovie`.
`startDrag`	Specifikuje movie klip jako tažitelný a začíná tažení movie klipu.
`stop`	Zastavuje aktuálně hrající animaci.
`stopDrag`	Zastavuje tažení jakéhokoliv movie klipu, který byl tažen.
`swapDepths`	Vyměňuje úroveň hloubky specifikovaného animace s animací na specifikované úrovni hloubky.
`unloadMovie`	Odstraňuje animaci natažené pomocí `loadMovie`.

MovieClip.attachMovie

Syntaxe

`jakykolivMovieClip.attachMovie`(idJméno, novéjméno, hloubka);

Argumenty

idJméno Jméno movie klipu z knihovny, který má být připojen. Toto je jméno vložené do pole Identifikátor v dialog boxu Symbol `Linkage Properties`.

novéjméno Unikátní jméno instance pro movie klip, který má být připojen.

hloubka Celé číslo určující úroveň hloubky, ve které je umístěna animace.

Popis

Metoda; vytváří novou instanci movie klipu v knihovně a připojuje ji k movie klipu určenému v `jakýkolivMovieKlip`. Pro odstranění movie klipu připojeného pomocí `attachMovie`, použijte akci nebo metodu `removeMovieClip`, nebo `unloadMovie`.

Přehrávač

Flash 5 nebo novější.

Viz také

`removeMovieClip`
`unloadMovie`
`MovieClip.removeMovieClip`
`MovieClip.unloadMovie`

MovieClip.duplicateMovieClip

Syntaxe

`jakýkolivMovieKlip.duplicateMovieClip`(novéjméno, hloubka);

Argumenty

novéjméno Unikátní identifikátor pro duplikování movie klipu.

hloubka Číslo určující úroveň hloubky, ve které je umístěna specifikovaná animace.

Popis

Metoda; vytváří instanci specifikovaného movie klipu, zatímco animace hraje. Duplikované movie klipy začínají vždy hrát na snímku 1, bez ohledu na kterém snímku je původní movie klip při volání metody `duplicateMovieClip`. Proměnné v rodičovském movie klipu nejsou kopírovány do duplikovaného movie klipu. Jestliže je rodičovský movie klip zrušen, duplikovaný movie klip je také zrušen. Movie klipy přidané pomocí duplicateMovieClip mohou být zrušeny pomocí akce nebo metody `removeMovieClip`.

Přehrávač

Flash 5 nebo novější.

Viz také

`removeMovieClip`
`MovieClip.removeMovieClip`

MovieClip.getBounds

Syntaxe

jakýkolivMovieKlip.`getBounds`(cílSouřadnicovýProstor);

Argumenty

cílSouřadnicovýProstor Cílová cesta Časové osy, jejíž souřadnicový prostor chcete použít jako referenční bod.

Popis

Metoda; udává minimální a maximální hodnoty souřadnic **x** a **y** Movie Klipu pro cílový souřadnicnový prostor určený v argumentu. Získaný objekt bude obsahovat vlastnosti `{xMin, xMax, yMin, xMax}`. Použijte metody objektu MovieClip `localToGlobal` a `globalToLocal` pro konverzi lokálních souřadnic movie klipu na souřadnice Scény nebo souřadnic Scény na lokální souřadnice.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad používá **getBounds** pro získání hraničního boxu instance **mujMovieKlip** v prostoru souřadnic hlavní animace:
```
mujMovieKlip.getBounds(._root);
```

- **Viz také**
  MovieClip.globalToLocal
  MovieClip.localToGlobal

MovieClip.getBytesLoaded

- **Syntaxe**
```
jakýkolivMovieKlip.getBytesLoaded();
```

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává počet natažených bytů pro specifikovaný objekt Movie Clip. Protože interní movie klipy se natahují automaticky, obdržený výsledek pro tuto metodu a **MovieClip.getBytesTotal** bude stejný, pokud specifikovaný objekt Movie Clip odkazuje na interní movie klip. Tato metoda je určena pro použití v natažených animacích, na určení toho, jaké procento externí animace bylo nataženo, můžete porovnat hodnotu **getBytesLoaded** s hodnotou **getBytesTotal**.

- **Přehrávač**
  Flash 5 nebo novější.

MovieClip.getBytesTotal

- **Syntaxe**
```
jakýkolivMovieklip.getBytesTotal();
```

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává velikost specifikovaného objektu Movie klip v bytech. Pro movie klipy, které jsou externí, (kořen movie nebo movie klip, který byl natažen do cíle nebo úrovně) je obdržená hodnota velikost SWF souboru.

- **Přehrávač**
  Flash 5 nebo novější.

MovieClip.getURL

- **Syntaxe**
```
jakýkolivMovieKlip.getURL( URL [, okno, proměnné] );
```

- **Argumenty**

URL — Url, ze kterého se získá dokument.

window (okno) — Volitelný argument určující jméno, snímek nebo výraz určující okno nebo HTML rám, do kterého má být natažen dokument. Můžete také použít jedno z následujících rezervovaných jmen cílů: **_self** specifikuje aktuální rám v aktuálních okně, **_blank** specifikuje nové okno, **_parent** specifikuje rodiče aktuálního rámu, **_top** specifikuje rám nejvyšší úrovně v aktuálním okně.

variables (proměnné) — Volitelný argument určující způsob pro posílání proměnných spojených s movie, které se natahuje. Jestliže zde nejsou žádné proměnné, tento argument vynechte; jinak určete, zda se mají proměnné natáhnout pomocí použití způsobu GET nebo POST. GET připojí proměnné na konec URL a je používán pro malé počty proměnných. POST pošle proměnné v odděleném HTTP a je používán pro dlouhé řetězce proměnných.

Popis

Metoda; natáhne dokument ze specifikovaného URL do specifikovaného okna. Metoda getURL může být použita také pro propuštění proměnných do jiné aplikace definované v URL použitím způsobu GET nebo POST.

Přehrávač

Flash 5 nebo novější.

MovieClip.globalToLocal

Syntaxe

```
jakýkolivMovieKlip.globalToLocal(point);
```

Argumenty

point Jméno nebo identifikátor objektu vytvořený pomocí generického objektu Object, určující souřadnice **x** a **y** jako vlastnosti.

Popis

Metoda; konvertuje objekt point ze souřadnic Scény (globálních souřadnic) na souřadnice movie klipu (na souřadnice lokální).

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad konvertuje globální souřadnice **x** a **y** objektu point na lokální souřadnice movie klipu:

```
onClipEvent(mouseMove){
      point = new Object();
      point.x = _root._xmouse;
      point.y = _root._ymouse;
      globalToLocal(point);
      _root.out = _xmouse + "===" + _ymouse;
      _root.out2 = point.x + "===" + point.y;
      updateAfterEvent();
}
```

Viz také

`MovieClip.localToGlobal`
`MovieClip.getBounds`

MovieClip.gotoAndPlay

Syntaxe

```
jakýkolivMovieKlip.gotoAndPlay(snímek);
```

Argumenty

snímek Číslo snímku, na který bude poslána hrací hlava.

Popis

Metoda; začíná hrát animaci na specifikovaném snímku.

Přehrávač

Flash 5 nebo novější.

MovieClip.gotoAndStop

Syntaxe

```
jakýkolivMovieKlip.gotoAndStop(snímek);
```

Argumenty

snímek Číslo snímku, na který bude poslána hrací hlava.

Popis

Metoda; zastavuje animaci na specifikovaném snímku.

Přehrávač

Flash 5 nebo novější.

MovieClip.hitTest

Syntaxe

```
jakýkolivMovieKlip.hitTest(x, y, shapeFlag);
jakýkolivMovieKlip.hitTest(target);
```

Argumenty

x Souřadnice **x** zasažené oblasti na Scéně.

y Souřadnice **y** zasažené oblasti na Scéně.

Souřadnice **x** a **y** jsou definovány v globálním souřadnicovém prostoru.

target (cíl) Cílová cesta zasažené oblasti, která může být protnuta nebo překryta instancí určenou v **jakýkolivMovieKlip**. Cíl obvykle reprezentuje tlačítko nebo textové pole.

shapeFlag Booleánská hodnota určující, zda ohodnotit celý tvar specifikované instance (`true`) nebo pouze hraničního boxu (`false`).
Tento argument může být specifikován pouze tehdy, jestliže je zasažená oblast identifikována pomocí použití argumentů souřadnice **x** a **y**.

Popis

Metoda; ohodnocuje instanci specifikovanou v **jakýkolivMovieKlip**, zda se překrývá nebo protíná se zasaženou oblastí identifikovanou v cíli nebo argumentech souřadnice **x** a **y**.

Použití 1 srovnává souřadnice **x** a **y** s tvarem nebo hraničním boxem specifikované instance podle nastavení `shapeFlag`. Jestliže je `shapeFlag` nastaven na `true`, je ohodnocena pouze oblast aktuálně okupovaná instancí na Scéně, a jestliže se **x** a **y** překrývají na jakémkoliv bodě, je obdržena hodnota `true`. Toto je užitečné pro určování, zda je movie klip uvnitř specifikované zasažené oblasti.

Použití 2 ohodnocuje hraniční boxy cíle a specifikované instance a udává `true`, jestliže se překrývají nebo protínají v jakémkoliv bodě.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá `hitTest` s vlastnostmi `x_mouse` a `y_mouse` pro určení, zda je myš na hraničním boxu cíle:

```
if ( hitTest(_root._xmouse, _root._ymouse, false));
```

Následující příklad používá `hitTest` pro určení, zda movie klip mic překrývá nebo se protíná s movie klipem ctverec:

```
if (_root.mic, hitTest (_root.ctverec)){
      trace(„míč protíná čtverec");
}
```

Viz také

`MovieClip.localToGlobal`
`MovieClip.globalToLocal`
`MovieClip.getBounds`

MovieClip.loadMovie

Syntaxe

```
jakýkolivMovieKlip.loadMovie(url [,proměnné ]);
```

Argumenty

url Absolutní nebo relativní URL pro SWF soubor, který má být natažen. Relativní cesta musí být relativní k SWF. URL musí být ve stejné subdoméně jako URL, kde je animace aktuálně umístěna. Pro použití ve Flash Přehrávači, nebo pro testování v režimu test movie v autorském prostředí Flashe, musí být všechny SWF soubory uloženy ve stejné složce a jména souboru nemohou zahrnovat specifikace složky nebo disku.

proměnné Volitelný argument určující způsob posílání proměnných spojených s animací, která má být natažena. Argument musí být řetězec „GET“ nebo „POST“. Jestliže zde nejsou žádné proměnné, tento argument vynechte; jinak určete, zda se mají proměnné natáhnout pomocí způsobu GET nebo POST. GET připojí proměnné na konec URL a je používán pro malá množství proměnných. POST posílá proměnné v odděleném HTTP a je používán pro dlouhé řetězce proměnných.

Popis

Metoda; přehrává dodatečné animace bez zavření Flash Přehrávače. Normálně Flash Přehrávač zobrazí jednu animaci (SWF soubor) a potom se zavře. Metoda loadMovie vám umožňuje zobrazit několik animací najednou nebo mezi nimi přepínat bez natahování dalšího HTML dokumentu.

Pro odstranění animací natažených pomocí akce **`loadMovie`**, použijte akci **`unloadMovie`**.

Použijte metodu **`loadVariables`** pro udržení aktivního movie a updatujte proměnné novými hodnotami.

Přehrávač

Flash 5 nebo novější.

Viz také

`MovieClip.loadVariables`
`MovieClip.unloadMovie`

MovieClip.loadVariables

Syntaxe

```
můjMovieKlip.loadVariables(url, proměnné);
```

Argumenty

url Absolutní nebo relativní URL pro externí soubor. Hostitel pro URL musí být ve stejné subdoméně jako movie klip.

proměnné Způsob pro získávání proměnných. GET připojí proměnné na konec URL a je používán pro malá množství proměnných. POST pošle proměnné v odděleném HTTP a je používán pro dlouhé řetězce proměnných.

Popis

Metoda; čte data z externího souboru a nastavuje hodnoty proměnných v animaci nebo movie klipu. Externí soubor může být textový soubor generovaný CGI scénářem, **`Active Server Pages`** (ASP) nebo PHP a může obsahovat jakýkoliv počet proměnných.

Tato metoda může být také použita pro updatování proměnných v aktivní animaci novými hodnotami.

Tato metoda požaduje, aby text v URL byl ve standardním MIME formátu: **`aplikace/x-www-urlformencoded (CGI script formát).`**

Přehrávač

Flash 5 nebo novější.

Viz také

`MovieClip.loadMovie`

MovieClip.localToGlobal

Syntaxe

```
jakýkolivMovieKlip.localToGlobal (point);
```

Argumenty

point Jméno nebo identifikátor objektu vytvořeného pomocí objektu Object, určujícího souřadnice **x** a **y** jako vlastnosti.

Popis

Metoda; konvertuje objekt point ze souřadnic movie klipu (lokálních) na souřadnice Scény (globální).

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad konvertuje souřadnice **x** a **y** objektu point ze souřadnic movie klipu (lokálních) na souřadnice Scény (globální). Lokální souřadnice **x** a **y** jsou určeny pomocí použití **xmouse** a **ymouse** pro získání souřadnic **x** a **y** pozice myši.

```
onClipEvent(mouseMove){
        point = new Object();
        point.x = _xmouse;
        point.y = _ymouse;
        _root.out3 = point.x + "===" + point.y;
        _root.out = _root._xmouse + "===" + _root._ymouse;
        localToGlobal (point);
        _root.out2 = point.x + "===" + point.y;
        updateAfterEvent();
}
```

Viz také

`MovieClip.globalToLocal`

MovieClip.nextFrame

Syntaxe

```
jakýkolivMovieKlip.nextFrame();
```

Argumenty

Žádné.

Popis

Metoda; posílá hrací hlavu na další snímek movie klipu.

Přehrávač

Flash 5 nebo novější.

MovieClip.play

Syntaxe

```
jakýkolivMovieKlip.play();
```

Argumenty

Žádné.

Popis

Metoda; hraje movie klip.

Přehrávač

Flash 5 nebo novější.

MovieClip.prevFrame

Syntaxe

```
jakýkolivMovieKlip.prevFrame();
```

Argumenty

Žádné.

Popis

Metoda; posílá hrací hlavu na předchozí snímek a zastavuje ji.

Přehrávač

Flash 5 nebo novější.

MovieClip.removeMovieClip

Syntaxe
```
jakýkolivMovieKlip.removeMovieClip();
```

Argumenty
Žádné.

Popis
Metoda; odstraňuje movie klip instanci vytvořenou pomocí akce **duplicateMovieClip** nebo metody objektu MovieClip **duplicateMovieClip** nebo **attachMovie**.

Přehrávač
Flash 5 nebo novější.

Viz také
MovieClip.loadMovie
MovieClip.attachMovie

MovieClip.startDrag

Syntaxe
jakýkolivMovieKlip.startDrag([lock, left, right, top, bottom]);

Argumenty
lock — Booleovská hodnota určující, zda je tažitelný movie klip zamknutý na střed pozice myši (**true**) nebo je zamknutý na bod, kde uživatel poprvé kliknul na movie klip (**false**). Tento argument je volitelný.

left, top, right, bottom — Hodnoty relativní k souřadnicím rodičovského movie klipu, které specifikují omezení obdélníku pro movie klip. Tyto argumenty jsou volitelné.

Popis
Metoda; umožňuje uživateli táhnout specifikovaný movie klip. Movie klip zůstává tažitelný, dokud není explicitně zastaven voláním metody **stopDrag** nebo dokud není učiněn tažitelným jiný movie klip. V jednom čase může být tažitelný pouze jeden movie klip.

Přehrávač
Flash 5 nebo novější.

Viz také
MovieClip.stopDrag
_droptarget

MovieClip.stop

Syntaxe
```
jakýkolivMovieKlip.stop();
```

Argumenty
Žádné.

Popis
Metoda; zastavuje movie klip, který aktuálně hraje.

Přehrávač
Flash 5 nebo novější.

MovieClip.stopDrag

Syntaxe
```
jakýkolivMovieKlip.stopDrag();
```

Argumenty
Žádné.

Popis
Metoda; končí akci tažení zavedenou metodou **startDrag**. Movie klip zůstává tažitelný, dokud není přidána metoda **stopDrag** nebo dokud se jiný movie klip nestane tažitelným. V jednom čase může být tažitelný pouze jeden movie klip.

Přehrávač
Flash 5 nebo novější.

Viz také
_droptarget
MovieClip.startDrag

MovieClip.swapDepths

Syntaxe

```
jakýkolivMovieKlip.swapDepths(hloubka);
jakýkolivMovieKlip. swapDepths (cíl);
```

Argument

cíl Instance movie klipu, jejíž hloubka bude vyměněna instancí určenou v **jakýkolivMovieKlip**. Obě instance musí mít stejný rodičovský movie klip.

hloubka Číslo určující úroveň hloubky, kam má být umístěn **jakýkolivMovieKlip**.

Popis

Metoda; vyměňuje navršení nebo pořadí (úroveň hloubky) specifikované instance s movie klipem specifikovaným argumentem cíl nebo s movie klipem, který je právě na úrovni hloubky specifikované v argumentu. Obě animace musí mít stejný rodičovský movie klip. Výměna úrovně hloubky movie klipů má efekt v tom, že jeden movie klip se posune před nebo za jiný. Jestliže je movie klip interpolován, pokud je tato metoda volána, interpolování je zastaveno.

Přehrávač

Flash 5 nebo novější.

Viz také

`_level`

MovieClip.unloadMovie

Syntaxe

```
jakýkolivMovieKlip.unloadMovie();
```

Argumenty

Žádné.

Popis

Metoda; odstraňuje movie klip natažený pomocí metody MovieClipu loadMovie nebo `attachMovie`.

Přehrávač

Flash 5 nebo novější.

Viz také

`MovieClip.loadMovie`
`MovieClip.attachMovie`

_name

Syntaxe

```
instancejméno._name
instancejméno._name = hodnota;
```

Argumenty

instancejméno Jméno instance movie klipu, pro který má být nastavena nebo získána vlastnost `_name`.

hodnota Řetězec, který specifikuje jméno nové instance.

Popis

Vlastnost; určuje jméno movie klip instance.

Přehrávač

Flash 4 nebo novější.

NaN

Syntaxe
```
NaN
```

Argumenty
Žádné.

Popis
Proměnná; předdefinovaná proměnná s hodnotou IEEE-754 pro **NaN** (Ne Číslo).

Přehrávač
Flash 5 nebo novější.

ne (not equal - string specific)

Syntaxe
výraz1 ne výraz2

Argumenty
výraz1,výraz2 Čísla, řetězce nebo proměnné.

Popis
Operátor (srovnání); porovnává **výraz1** v **výrazem2** a udává true, jestliže **výraz1** není roven **výrazu2**; jinak udává false.

Přehrávač
Flash 4 nebo novější. Tento operátor byl zavržen ve Flash 5; je doporučeno používat nový operátor **!= (není rovno)**.

Viz také
```
!= (inequality)
```


new

Syntaxe
```
new konstruktor();
```

Argumenty
konstruktor Funkce následovaná volitelnými argumenty v závorkách. Funkce je obvykle jméno typu objektu **(Například Array, Math, Number, Object)**, který má být konstruován.

Popis
Operátor; vytváří nový, původně anonymní objekt, volá funkci identifikovanou argumentem konstruktor, vkládá volitelné argumenty v závorkách a expeduje nově vytvořený objekt jako hodnotu klíčového slova **this**. Konstrukční funkce potom může **this** použít pro konkretizaci nového objektu.

prototyp**_vlastnost** konstrukční funkce objektu je kopírována do **_proto_vlastnosti** nového objektu. Výsledkem je to, že nový objekt podporuje všechny metody a vlastnosti určené v konstrukční funkci objektu **Prototyp**.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad vytváří objekty **kniha1** a **kniha2** pomocí použití operátoru **new**.
```
function Kniha(jmeno, cena)
{
       this.jmeno = jmeno;
       this.cena = cena;
}
kniha1 = new Kniha („Konfederace hlupáků“, 19.95);
kniha2 = new Kniha („The Floating Opera“,10.95);
```

Viz také
```
[] (array access operator)
{} (object initializer)
```

Část konstrukční metody uvnitř úvodu objektu.

newline

Syntaxe
```
newline;
```

Argumety
Žádné.

Popis
Konstanta; vkládá nový řádek znak **({)** vložením prázdného řádku do kódu ActionScript. Použijte newline pro vytvoření prostoru pro informaci, která je získávána funkcí nebo akcí ve vašem kódu.

Přehrávač
Flash 4 nebo novější.

nextFrame

Syntaxe
```
nextFrame();
```

Argumenty
Žádné.

Popis
Akce; posílá hrací hlavu na další snímek a zastavuje ji.

Přehrávač
Flash 2 nebo novější.

Příklad
Když uživatel klikne na tlačítko, ke kterému je připojena akce **nextFrame**, je hrací hlava poslána na další snímek.
```
on(release){
        nextFrame(5);
}
```


nextScene

Syntaxe
```
nextScene();
```

Argumenty
Žádné.

Popis
Akce; posílá hrací hlavu na **snímek 1** další scény a zastavuje ji.

Přehrávač
Flash 2 nebo novější.

Příklad
Tato akce je připojena k tlačítku, které když je stisknuto a uvolněno posílá hrací hlavu na **snímek 1** další scény.
```
on(release) {
        nextScene();
}
```

not

Syntaxe
```
not výraz
```

Argumenty
výraz Jakákoliv proměnná nebo jiný výraz, který se konvertuje na Booleovskou hodnotu.

Popis
Operátor; vykonává logickou operaci NOT ve Flash 4 Přehrávači.

Přehrávač
Flash 4 nebo novější. Tento operátor byl ve Flash 5 zavržen; je doporučeno používat nový operátor ! (logical NOT).

Viz také
```
! (logical NOT)
```

null

Syntaxe
```
null
```

Argumenty
Žádné.

Popis
Klíčové slovo; speciální hodnota, která může být připojena k proměnným nebo získána funkcí, jestliže nebyly poskytnuty žádná data. Můžete použít **null** pro reprezentaci hodnot, které chybí nebo nemají definovaný typ dat.

Přehrávač
Flash 5 nebo novější.

Příklad
V numerickém kontextu se **null** ohodnocuje na 0. S **null** mohou být provedeny testy rovnosti. V této zprávě nemá binarický node strom žádné dítě, takže pole pro jeho levé dítě může být nastaveno na **null**.

```
if(strom.levy == null) {
       strom.levy = new StromNode();
}
```

Number (function)

Syntaxe
```
Number(výraz);
```

Argumenty

výraz	Řetězec, Booleovská nebo jiný výraz, který se má konvertovat na číslo.

Popis
Funkce; konvertuje argument **x** na číslo a udává hodnotu následovně:
Jestliže je **x** číslo, udaná hodnota je **x**.

Jestliže je **x** Booleovská, udaná hodnota je 1, jestliže je **x** **true** , 0 jestliže je **x** **false**.

Jestliže je **x** řetězec, funkce přistupuje k analýze **x** jako decimálnímu číslu s volitelným stopovacím exponentem, to je 1.57505e-3.

Jestliže není **x** definováno, udaná hodnota je 0.

Tato funkce je použita pro konverzi Flash 4 souborů obsahujících zavržené operátory, které jsou importovány do Flash 5 autorského prostředí. Pro více informací se podívejte na operátor &.

Přehrávač
Flash 4 nebo novější.

Viz také
Number (object)

Number (object)
Objekt Number je jednoduchý obalový objekt pro číselný typ dat, to znamená, že můžete manipulovat s původními numerickými hodnotami pomocí použití metod a vlastností spojených s objektem Number. Funkčnost poskytnutá tímto objektem je stejná jako u JavaScript objektu Number.

Konstruktor Number musíte použít když voláte metody objektu Number, konstruktor ale nemusíte použít, když voláte vlastnosti objektu Number. Následující příklady specifikují syntaxi pro volání metod a vlastností objektu Number:

Toto je příklad volání metody toString objektu Number:
```
myNumber = new Number (1234);
myNumber.toString();
```

Udává řetězec obsahující binarickou reprezentaci čísla 1234.

Toto je příklad volání vlastnosti MIN_VALUE (také nazvaná konstanta) objektu Number:
```
nejmensi = Number.MIN_VALUE
```

Přehled metod objektu Number

Metoda	Popis
toString	Udává řetězcovou reprezentaci objektu Number.
valueOf	Udává původní hodnotu objektu Number.

Přehled vlastností objektu Number

Metoda	Popis
`MAX_VALUE`	Konstanta reprezentující největší reprezentovatelné číslo (dvojitá-přesnost IEEE-754). Toto číslo je přibližně 1.7976931348623158e+308.
`MIN_VALUE`	Konstanta reprezentující nejmenší reprezentovatelné číslo (dvojitá-přesnost IEEE-754). Toto číslo je přibližně 5e-324.
`NaN`	Konstanta reprezentující hodnotu pro Not Number (NaN).
`NEGATIVE_INFINITY`	Konstanta reprezentující hodnotu pro negativní nekonečno.
`POSITIVE_INFINITY`	Konstanta reprezentující hodnotu pozitivního nekonečna. Tato hodnota je stejná jako všeobecná proměnné Nekonečno.

Konstruktor pro objekt Number

Syntaxe
```
myNumber = new Number(hodnota);
```

Argumenty
hodnota Numerická hodnota objektu Number, který je vytvářen nebo hodnota, která má být konvertována na číslo.

Popis
Konstruktor; vytváří nový objekt Number. Musíte použít konstruktor Number, když používáte metody objektu Number `toString` a `valueOf`. Nepoužívejte konstruktor, když používáte vlastnosti objektu Number. Konstruktor new Number je používán především jako držitel místa. Instance objektu Number není stejná jako funkce Number, která konvertuje argument na původní hodnotu.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující kód konstruuje nové objekty Number:
```
n1 = new Number(3.4);
n2 = new Number(-10);
```

Viz také
`Number (function)`

Number.MAX_VALUE

Syntaxe
```
Number.MAX_VALUE
```

Argumenty
Žádné.

Popis
Vlastnost; největší reprezentovatelné číslo (dvojitá-přesnost IEEE-754). Toto číslo je přibližně 1.79E+308.

Přehrávač
Flash 5 nebo novější.

Number.MIN_VALUE

Syntaxe
```
Number.MIN_VALUE
```

Argumenty
Žádné.

Popis
Vlastnost; nejmenší reprezentovatelné číslo (dvojitá-přesnost IEEE-754). Toto číslo je přibližně 5e-324.

Přehrávač
Flash 5 nebo novější.

Number.NaN

- **Syntaxe**
  `Number.NaN`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; hodnota IEEE-754 reprezentující Not Number ().

- **Přehrávač**
  Flash 5 nebo novější.

Number.NEGATIVE_INFINITY

- **Syntaxe**
  `Number.NEGATIVE_INFINITY`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; udává IEEE-754 hodnotu reprezentující negativní nekonečno. Tato hodnota je stejná jako všeobecná proměnná Infinity (Nekonečno).

  Negativní nekonečno je speciální numerická hodnota, která je obdržena, když matematická operace nebo funkce, udá zápornou hodnotu vetší než může být reprezentována.

- **Přehrávač**
  Flash 5 nebo novější.

Number.POSITIVE_INFINITY

- **Syntaxe**
  `Number.POZITIVE_INFINITY`

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; udává IEEE-754 hodnotu reprezentující kladné nekonečno. Tato hodnota je stejná jako všeobecná proměnné Infinity (Nekonečno).

  Pozitivní nekonečno je speciální numerická hodnota, která je obdržena, když matematická operace nebo funkce, udá hodnotu větší než může být reprezentována.

- **Přehrávač**
  Flash 5 nebo novější.

Number.toString

- **Syntaxe**
  ```
  mojeCislo.toString(radix);
  ```

- **Argumenty**

  radix — Specifikuje numerický základ (od 2 do 36) pro použití pro konverzi číslo-na-řetězec. Jestliže nespecifikujete argument radix, nastavená hodnota je 10.

- **Popis**
  Metoda; udává řetězcovou reprezentaci specifikovaného objektu Number **(myNumber)**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklad používá metodu Number.toString, určující 2 za argument radix:
  ```
  myNumber = new Number (1000);
  (1000).toString(2);
  ```

  Udává řetězec obsahující binarickou reprezentaci čísla 1000.

Number.valueOf

Syntaxe
```
myNumber.valueOf();
```

Argumenty
Žádné.

Popis
Metoda; udává původní typ hodnoty specifikovaného objektu Number a konvertuje obalový objekt Number na původní typ hodnoty.

Přehrávač
Flash 5 nebo novější.

Object (object)
Generický objekt Object je v kořenu ActionScript hierarchie tříd. Funkčnost generického objektu Object je malou podmnožinou toho, co poskytuje objekt JavaScript Object.
Generický objekt Object požaduje Flash 5 Přehrávač.

Přehled metod objektu Object

Konstruktor pro objekt Object

Syntaxe
```
new Object();
new Object(hodnota);
```

Argumenty
hodnota Číslo, Booleovská nebo řetězec, který má být konvertován na objekt. Tento argument je volitelný. Jestliže nespecifikujete hodnotu, konstruktor vytvoří nový objekt bez definovaných vlastností.

Popis
Konstruktor; vytváří nový objekt Object.

Přehrávač
Flash 5 nebo novější.

Viz také
```
Sound.setTransform
Color.setTransform
```


Object.toString

Syntaxe
```
myObject.toString();
```

Argumenty
Žádné.

Popis
Metoda; konvertuje specifikovaný objekt na řetězec a udává ho.

Přehrávač
Flash 5 nebo novější.

Object.valueOf

Syntaxe
```
myObject.valueOf();
```

Argumenty
Žádné.

Popis
Metoda; udává původní hodnotu specifikovaného objektu. Jestliže objekt nemá původní hodnotu, je obdržen sám objekt.

Přehrávač
Flash 5 nebo novější.

onClipEvent

Syntaxe

```
onClipEvent(movieUdálost);{
...
}
```

Argumenty

movieUdálost je spouštěcí událost, která vykoná akce připojeny k movie klip instanci. Jakékoliv následující hodnoty mohou být specifikovány pro argument **movieUdálost**:

load Akce je inicializována, jakmile je movie klip konkretizován a objeví se na Časové ose.

unload Akce je inicializována na prvním snímku poté co je movie klip odstraněn z Časové osy. Akce spojené s movie klip událostí Unload jsou provedeny před jakýmikoliv akcemi připojenými k ovlivněnému snímku.

enterFrame Akce je inicializována, když hraje každý snímek, podobně jako akce připojené k movie klipu. Akce spojené s movie klip událostí **OnEnterFrame** jsou vykonány po jakýchkoliv akcích, které jsou připojeny k ovlivněným snímkům.

mouseMove Akce je inicializována pokaždé, když se hne myší. Použijte vlastnosti **_xmouse** a **_ymouse** pro určení aktuální pozice myši.

mouseDown Akce je inicializována, když je stisknuto levé tlačítko myši.

mouseUp Akce je inicializována, když je uvolněno levé tlačítko myši.

keyDown Akce je inicializována, když je stisknuta klávesa. Použijte metodu **Key.getCode**, pro získání informace o naposledy stisknuté klávese.

keyUp Akce je inicializována, když je klávesa uvolněna. Použijte metodu **Key.getCode**, pro získání informace o naposledy stisknuté klávese.

data Akce je inicializována, když je obdržen údaj a akci loadVariables nebo loadMovie. Pokud je specifikována akce pomocí loadVariables, událost data se objeví pouze jednou, když je natažena poslední proměnná. Když je specifikována akce pomocí loadMovie, událost data se objeví opakovaně pokaždé, když je získána každá část dat.

Popis

Ovladač; spouští akce definované pro specifickou instanci movie klipu.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující zpráva zahrnuje skript z externího souboru, když je instance movie klipu natažena a poprvé se objeví na Časové ose:

```
onClipEvent(load){
      #include „můjScénář.as“
}
```

Následující příklad používá **onClipEvent** s událostí **keyDown**. Událost **keyDown** je obvykle používána ve spojení s jedním nebo více metodami a vlastnostmi spojenými s objektem **Key**. Ve scénáři dole je **Key.getCode** použit pro zjištění, kterou klávesu uživatel stiskl; obdržená hodnota je spojena s vlastnostmi objektu Klávesa RIGHT nebo LEFT, a animace je řízena podle toho.

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

Následující příklad používá **onClipEvent** s událostí **mouseMove**. Vlastnosti **xmouse** a **ymouse** sledují pozici myši.

```
onClipEvent (mouseMove){
plochaX = _root.xmouse;
plochaY = _root.ymouse;
}
```

Viz také

```
on(mouseEvent)
Key (object)
_xmouse
_ymouse
```

on(mouseEvent)

Syntaxe

```
on(mouseEvent){
příkaz;
}
```

Argumenty

příkaz Instrukce pro vykonání, když se stane mouseEvent.

Akce **mouseEvent** může mít jeden z následujících argumentů:

press Tlačítko myši je stisknuto, zatímco ukazatel je na tlačítku.

release Tlačítko myši je uvolněno, zatímco ukazatel je na tlačítku.

releaseOutside Tlačítko myši je uvolněno, zatímco ukazatel je mimo tlačítka.

rollOver Ukazatel myši roluje na tlačítku.

rollOut Ukazatel myši roluje mimo oblast tlačítka.

dragOver Zatímco je ukazatel na tlačítku, bylo stisknuto tlačítko myši, když rolovalo mimo tlačítko a potom rolovalo zpět na tlačítko.

dragOut Zatímco je ukazatel na tlačítku, tlačítko myši je stisknuto a potom roluje mimo oblast tlačítka.

keyPress
(„klávesa“) Je stisknuta specifikovaná klávesa. Část argumentu klávesa je specifikována použitím klávesových kódů zapsaných v Dodatku B, „Klávesy Klávesnice a Hodnoty Kódů Kláves“, nebo jakýchkoliv konstant kódů zapsaných v přehledu Vlastností pro objekt Klávesa.

Popis

Ovladač; specifikuje událost myši nebo stisknutí klávesy, které spouští akci.

Přehrávač

Flash 2 nebo novější.

Příklad

V následujícím skriptu se vykoná akce **startDrag** když je myš stisknuta, a podmínkový příkaz je vykonán když je myš uvolněna a objekt je puštěn:

```
on(press){
      startDrag("kralik");
}
on(release){
      if(getproperty("", _droptarget) == target){
            setProperty ("kralik", _x, _root.kralik_x);
            setProperty ("kralik", _y, _root.kralik_y);
      }else {
_root.kralik_x = getProperty("kralik", _x);
_root.kralik_y = getProperty("kralik", _y);
_root.target = "pastvina";
      }
      trace(_root.kralik_y);
      trace(_root.kralik_x);
      stopDrag();
}
```

Viz také

`Key (objekt)`
`onClipEvent`

or

Syntaxe

podmínka1 `or` **podmínka2**

Argumenty

podmínka1,2 Výraz, který se ohodnocuje na **true** nebo false.

Popis

Operátor; odhodnocuje **podmínku1** a **podmínku2** a jestliže je jeden výraz **true**, potom je celý výraz **true**.

Přehrávač

Flash 4 nebo novější. Tento operátor byl ve Flash 5 zavržen a je doporučeno používat nový operátor `||`.

Viz také

`||` `(OR)`

ord

Syntaxe
```
ord(znak);
```

Argumenty
znak Znak, který se má konvertovat na číslo ASCII kódu.

Popis
Řetězcová funkce; konvertuje znaky na čísla ASCII kódu.

Přehrávač
Flash 4 nebo novější. Tato funkce byla ve Flash 5 zavržena a je doporučeno používat metody a vlastnosti objektu String.

Viz také
```
String (object)
```

_parent

Syntaxe
```
_parent.vlastnost = x
_parent._parent.vlastnost = x
```

Argumenty
vlastnost Vlastnost, která je specifikována pro aktuální a rodičovský movie klip.

x Hodnota nastavená pro vlastnost. Toto je volitelný argument nemusí být nastaven, závisí na vlastnosti.

Popis
Vlastnost; specifikuje nebo udává odkaz na movie klip, který obsahuje aktuální movie klip. Aktuální movie klip je movie klip obsahující aktuálně vykonávaný skript. Použijte `_parent` pro specifikaci relativní cesty.

Přehrávač
Flash 4 nebo novější.

Příklad
V následujícím příkladě je movie klip tabule dítětem movie klipu **trida**. Když se skript dole vykoná uvnitř movie klipu tabule, hrací hlava skočí na snímek 10 v Časové ose movie klipu trida.
```
_parent.gotoAndStop(10);
```

Viz také
```
_root
targetPath
```

parseFloat

Syntaxe
```
parseFloat(řetězec);
```

Argumenty
řetězec Řetězec, který se má analyzovat a konvertovat na plynoucí číslo.

Popis
Funkce; konvertuje řetězec na plynoucí číslo. Funkce analyzuje a udává čísla v řetězci, dokud analyzátor nedosáhne znaku, který není částí původního čísla. Jestliže řetězec nezačíná číslem, které může být analyzováno, `parseFloat` udá `NaN` nebo `0`. Bílý prostor před platnými celými čísly je ignorován, stejně jako nenumerické znaky.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující jsou příklady použití parseFloat pro ohodnocení různých typů čísel:
```
parseFloat(„-2“) udá -2
parseFloat(„2.5“) udá 2.5
parseFloat(„3.5e6“) udá 3.5e6 nebo 3500000
parseFloat(„foosloupec“) udá NaN
```

parseInt

Syntaxe

```
parseInt(výraz,radix);
```

Argumenty

výraz Řetězec, plynoucí číslo nebo jiný výraz, který se má analyzovat a konvertovat na celé číslo.

radix Celé číslo reprezentující radix (základ) čísla pro analyzování. Legální hodnoty jsou od 2 do 36. Tento argument je volitelný.

Popis

Funkce; konvertuje řetězec na celé číslo. Jestliže nemůže být specifikovaný řetězec v argumentech konvertovaný na číslo, funkce udá **`NaN`** nebo **`0`**. Celá čísla začínající s **`0`** nebo specifikující **`radix 8`** jsou interpretovány jako oktal čísla. Celá čísla začínající **`0x`** jsou interpretována jako hexadecimální čísla. Bílý prostor před platnými celými čísly je ignorován, stejně jako nenumerické znaky.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující jsou příklady použití parseInt pro ohodnocení různých typů čísel:

```
parseInt(„3.5“) udá 3.5
parseInt („sloupec“) udá NaN
parseInt („4foo“) udá 4
```

Hexadecimální konverze:

```
parseInt (“0x3F8“) udá 1016
parseInt („3E8“, 16) udá 1000
```

Binarická konverze:
parseInt („1010“, 2) udá 10 (decimální reprezentace binarického 1010)

Analyzování oktal čísla (v tomto případě je oktální číslo identifikováno pomocí radix,8):
parseInt („777“, 8) udá 511 (decimální reprezentace oktálního 777)

play

Syntaxe

```
play();
```

Argumenty

Žádné.

Popis

Akce; posunuje hrací hlavu dopředu na Časové ose.

Přehrávač

Flash 2 nebo novější.

Příklad

Následující kód používá příkaz **`if`** pro kontrolu hodnoty jména, které uživatel vloží. Jestliže uživatel vloží **`Steve`**, je volána akce **`play`** a hrací hlava se posune dopředu na Časové ose. Jestliže uživatel vloží cokoliv jiného než **`Steve`**, movie nehraje a je zobrazeno textové pole s proměnnou jménem varovani.

```
stop();
if(jmeno = “Steve“){
      play();
}else{
      varování = „Ty nejsi Steve!“;
}
```

prevFrame

Syntaxe

```
prevFrame();
```

Argumenty

Žádné.

Popis

Akce; posílá hrací hlavu na předchozí snímek a zastavuje ji.

Přehrávač

Flash 2 nebo novější.

Příklad

Když uživatel klikne na tlačítko, ke kterému je připojena akce prevFrame, je hrací hlava poslána na předchozí snímek.

```
on(release){
        prevFrame(5);
}
```

Viz také

`MovieClip.prevFrame`

prevScene

Syntaxe

```
prevScene();
```

Argumenty

Žádné.

Popis

Akce; posílá hrací hlavu na snímek 1 předchozí scény a zastavuje ji.

Přehrávač

Flash 2 nebo novější.

Viz také

`nextScene`

print

Syntaxe

```
print (cíl,"bmovie");
print (cíl,"bmax");
print (cíl,"bframe");
```

Argumenty

cíl Jméno instance movie klipu, který se má tisknout. V nastavení jsou tisknuty všechny snímky v animaci. Jestliže chcete tisknout pouze určité snímky, označte snímky pro tisk připojením popisku snímku **#P** k těmto snímkům v autorském prostředí.

bmovie Označuje hraniční box specifického snímku v animaci jako oblast tisku pro všechny tisknutelné snímky. Připojte popisek **#b** (v autorském prostředí) pro označení snímku, jehož hraniční box chcete použít jako oblast tisku.

bmax Označuje složeninu všech hraničních boxů, všech tisknutelných snímků, jako oblast tisku. Specifikujte argument **bmax**, pokud se tisknutelné snímky ve vaší animaci liší velikostí.

bframe Označuje hraniční **box** pro každý tisknutelný snímek, který má být použitý jako oblast tisku pro tento snímek. Toto změní oblast tisku pro každý snímek a měří objekty, aby padly oblasti tisku. Použijte **bframe**, jestliže máte objekty různých velikostí v každém snímku a chcete, aby každý objekt vyplnil tisknutou stránku.

Popis

Akce; tiskne movie klip cíl v souladu s modifikátorem tiskárny specifikovaným v argumentu. Jestliže chcete tisknout pouze určité snímky v cílové animaci, připojte popisek snímku **#P** ke snímkům, které chcete tisknout. Ačkoliv se akce print odrazí ve vyšší kvalitě tisků, než akce **printAsBitmap**, nemůže být použita pro tisk animací, které používají alfa průhlednosti nebo speciální barevné efekty.

Jestliže nespecifikujete argument oblast tisku, je oblast tisku v nastavení určena velikostí Scény natažené animace. Animace nedědí velikost hlavní Scény. Oblast tisku můžete ovládat určením argumentů **bmovie**, **bmax** nebo **bframe**.
Všechny tisknutelné prvky v animaci musí být zcela nataženy dříve než začne tisk.

Flash Přehrávač tisk podporuje tiskárny **PostScript** a **non-PostScript**. **Non-PostScrip**t tiskárny konvertují vektory na bitmapy.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad bude tisknout všechny tisknutelné snímky v **mojeMovie** s oblastí tisku definovanou hraničním boxem snímku s připojeným popiskem snímku **#b**:

```
print(„mojeMovie","bmovie");
```

Následující příklad bude tisknout všechny tisknutelné snímky v **mojeMovie** s oblastí tisku definovanou hraničním boxem každého snímku:

```
print(„mojeMovie","bframe");
```

Viz také

`printAsBitmap`

printAsBitmap

Syntaxe

```
printAsBitmap (cíl,"bmovie");
printAsBitmap (cíl,"bmax");
printAsBitmap (cíl,"bframe");
```

Argumenty

cíl Jméno instance movie klipu, která se má tisknout. V nastavení jsou tisknuty všechny snímky. Jestliže chcete tisknout pouze určité snímky, označte snímky pro tisk připojením popisku snímku **#P**.

bmovie Označuje hraniční box specifického snímku v animaci jako oblast tisku pro všechny tisknutelné snímky. Připojte popisek **#b** (v autorském prostředí) pro označení snímku, jehož hraniční box chcete použít jako oblast tisku.

bmax Označuje složeninu všech hraničních boxů, všech tisknutelných snímků, jako oblast tisku. Určete argument bmax, když se tisknutelné snímky ve vaší animaci liší velikostí.

bframe Označuje hraniční box pro každý tisknutelný snímek, který má být použitý jako oblast tisku pro tento snímek. Toto změní oblast tisku pro každý snímek a měří objekty, aby padly oblasti tisku. Použijte **bframe**, jestliže máte objekty různých velikostí v každém snímku a chcete, aby každý objekt vyplnil tisknutou stránku.

Popis

Akce; tiskne movie klip cíl jako bitmapu. Použijte **printAsBitmap** pro tisk animací, které obsahují snímky s objekty, které používají průhlednost nebo barevné efekty. Akce **printAsBitmap** tiskne v nejvyšším dostupném rozlišení tiskárny, aby zvládla tolik ostrosti a kvality, kolik je možné. Pro kalkulaci tisknutelné velikosti souboru snímku označeného pro tisk jako bitmapa násobte šířku v pixelech výškou v pixelech a rozlišením tiskárny.

Jestliže vaše animace neobsahuje **alfa** průhlednosti nebo barevné efekty, je doporučováno, abyste používali akci print pro kvalitnější výsledky.

V nastavení je oblast tisku určena velikostí Scény natažené animace. Animace nedědí velikost hlavní Scény. Oblast tisku můžete ovládat určením argumentů **bmovie**, **bmax** nebo **bframe**.

Všechny tisknutelné prvky v animaci musí být zcela nataženy dříve než začne tisk. Flash Přehrávač tisk podporuje tiskárny **PostScript** a **non-PostScript**. Non-PostScript tiskárny konvertují vektory na bitmapy.

Přehrávač

Flash 5 nebo novější.

Viz také

print

_quality

Syntaxe

```
_quality
_quality = x;
```

Argumenty

x Řetězec specifikující jednu z následujících hodnot:

LOW Předložení nízké kvality. Grafiky nejsou antialiazed(vyrovnány), bitmapy nejsou vyhlazené.

MEDIUM Předložení střední kvality. Grafiky jsou antialiased použitím mřížky 2x2, ale bitmapy nejsou vyhlazeny. Vhodné pro movies, které neobsahují text.

HIGH Předložení vysoké kvality. Grafiky jsou antialiased použitím mřížky 4x4 a bitmapy jsou vyhlazeny, jestliže je movie statické. Toto je nastavené předložení nastavení kvality použité Flashem.

BEST Předložení velmi vysoké kvality. Grafiky jsou **antialiased** použitím mřížky 4x4 a bitmapy jsou vždy vyhlazené.

Popis
Vlastnost (všeobecná); nastavuje nebo získává předloženou kvalitu použitou pro animaci.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad nastavuje kvalitu zobrazení staraKvalita na HIGH:

```
staraKvalita =_quality
_quality = „HIGH“;
```

Viz také
`_highquality`

random

Syntaxe
```
random();
```

Argumenty
hodnota Nejvyšší celé číslo pro které náhodné udá hodnotu.

Popis
Funkce; udává náhodné celé číslo mezi 0 a celým číslem určeným v argumentu hodnota.

Přehrávač
Flash 4. Tato funkce je zavržena ve Flash 5; je doporučeno používat metodu `Math.random`.

Příklad
Následující použití random udává hodnotu 0,1,2,3 nebo 4:

```
random(5);
```

Viz také
`Math.random`

removeMovieClip

Syntaxe
```
removeMovieClip(cíl);
```

Argumenty
cíl Cílová cesta movie klip instance vytvořené pomocí `duplicateMovieClip` nebo jméno instance movie klipu vytvořené pomocí metod objektu `MovieClip attachMovie` nebo `duplicateMovie`.

Popis
Akce; ruší movie klip instanci, která byla vytvořena pomocí metod objektu Movie klip `attachMovie` nebo `duplicateMovieClip` nebo pomocí akce `duplicateMovieClip`.

Přehrávač
Flash 4 nebo novější.

Viz také
`duplicateMovieClip`
`MovieClip.duplicateMovieClip`
`MovieClip.attachMovie`
`MovieClip.removeMovieClip`

return

Syntaxe
```
return [výraz];
return;
```

Argumenty
výraz Typ, řetězec, číslo, pole nebo objekt, který se má ohodnotit a obdržet jako hodnota funkce. Tento argument je volitelný.

Popis
Akce; specifikuje hodnotu obdrženou funkcí. Když je obdržená akce vykonána, výraz je ohodnocen a obdržen jako hodnota funkce. Obdržená akce způsobí, že se zastaví vykonávání akce. Jestliže příkaz `return` je použit samostatně nebo jestliže Flash nenarazí na příkaz return během akce smyčkování, udá `null`.

Přehrávač
Flash 5 nebo novější.

Příklad

Následující je příkladem použití return:

```
function sum(a,b,c){
return a + b + c;
}
```

Viz také

`function`

_root

Syntaxe

```
_root;
_root.movieKlip;
_root.akce;
```

Argumenty

movieKlip Jméno instance movie klipu.

akce Hodnota nastavená pro vlastnost. Toto je volitelný argument a nemusí být třeba ho nastavovat - závisí na vlastnosti.

Popis

Vlastnost; specifikuje nebo udává odkaz na kořenovou Časovou osu animace. Jestliže má animace několik úrovní, kořenová Časová osa je na úrovni obsahující aktuálně vykonávaný skript. Například, jestliže skript na **úrovni 1** ohodnocuje `_root`, je obdržena **úroveň 1**.

Specifikování `_root` je stejné jako použití slash (zpětné lomítko) označení (`/`) pro určení absolutní cesty uvnitř aktuální úrovně.

Přehrávač

Flash 4 nebo novější.

Příklad

Následující příklad zastavuje Časovou osu úrovně obsahující aktuálně vykonávaný skript:

```
_root1.stop();
```

Následující příklad posílá Časovou osu v aktuální úrovni na snímek 3:

```
_root.gotoAndStop(3);
```

Viz také

`_parent`
`targetPath`

_rotation

Syntaxe

```
instancejméno._rotation
instancejméno._rotation = celéčíslo
```

Argumenty

celé číslo Počet stupňů, o které se má movie klip otočit.

instancejméno Movie klip, který má rotovat.

Popis

Vlastnost; určuje rotaci movie klipu ve stupních.

Přehrávač

Flash 4 nebo novější.

scroll

Syntaxe

```
proměnná_jméno.scroll = x
```

Argumenty

proměnná_jméno Jméno proměnné spojené s textovým polem.

x Číslo nejvýše viditelného řádku v textovém poli. Můžete specifikovat tuto hodnotu nebo použít nastavenou **hodnotu 1**. Flash Přehrávač updatuje tuto hodnotu, jakmile uživatel roluje nahoru a dolů textové pole.

Popis

Vlastnost; kontroluje zobrazení informace v textovém poli spojeném s proměnnou. Vlastnost **scroll** definuje, kde v poli začíná zobrazení obsahu; poté co je nastavena, Flash Přehrávač ji updatuje s tím, jak uživatel roluje skrz textové pole. Vlastnost **scroll** je užitečná pro řízení uživatelů na určitý odstavec v dlouhé pasáži textu, nebo vytváření rolujících textových polí. Tato vlastnost může být získána a modifikována.

Přehrávač

Flash 4 nebo novější.

Viz také

maxscroll

Selection (object)

Objekt Selection (Výběr) umožňuje nastavit a kontrolovat aktuálně zaměřené editovatelné textové pole. Aktuálně zaměřené editovatelné textové pole je pole, kde je aktuálně umístěný uživatelův kurzor myši. Indexy výběr-rozsah mají základ nula (kde je první pozice **0**, druhá pozice je **1** atd.).
Pro objekt Selection není žádná konstrukční metoda, protože může být aktuálně zaměřeno pouze jedno pole.

Přehled metod objektu Selection

Metoda	Popis
getBeginIndex	Udává index na začátku rozsahu výběru - 1, jestliže není žádný index nebo vybrané pole.
getCaretIndex	Udává aktuální pozici caret (šipka nahoru, vynechávka) v aktuálně zaměřeném rozsahu výběru -1, jestliže není žádná pozice caret nebo aktuálně zaměřený rozsah výběru.
getEndIndex	Udává index na konci rozsahu výběru -1, jestliže není žádný index nebo aktuálně vybrané pole.
getFocus	Udává jméno proměnné pro aktuálně zaměřené editovatelné textové pole. Udává nula, jestliže zde není aktuálně zaměřené editovatelné textové pole.
setFocus	Zaměřuje editovatelný text spojený s proměnnou specifikovanou v argumentu.
setSelection	Nastavuje počáteční a konečné indexy rozsahu výběru.

Selection.getbeginIndex

Syntaxe

```
Selection.getBeginIndex();
```

Argumenty

Žádné.

Popis

Metoda; udává index na začátku rozsahu výběru. Jestliže žádný index neexistuje, nebo žádné pole nemá aktuálně střed, metoda udá -1. Indexy rozsahu výběru mají základ nula (kde první pozice je 0, druhá pozice je 1, atd.).

Přehrávač

Flash 5 nebo novější.

Selection.getCaretIndex

Syntaxe

```
Selection.getCaretIndex();
```

Argumenty

Žádné.

Popis

Metoda; udává index pozice blikajícího kurzoru. Jestliže není žádný blikající ukazatel myši zobrazen, metoda udá -1. Indexy rozsahu výběru mají základ nula (kde první pozice je 0, druhá pozice je 1, atd.).

Přehrávač

Flash 5 nebo novější.

Selection.getEndIndex

Syntaxe

```
Selection.getEndIndex();
```

Argumenty

Žádné.

Popis

Metoda; udává konečný index aktuálně zaměřeného rozsahu výběru. Jestliže neexistuje žádný index nebo jestliže zde není žádný aktuálně zaměřený rozsah výběru, metoda udá -1. Indexy rozsahu výběru mají základ nula (kde první pozice je 0, druhá pozice je 1, atd.).

Přehrávač

Flash 5 nebo novější.

Selection.getFocus

Syntaxe

```
Selection.getFocus();
```

Argumenty

Žádné.

Popis

Metoda; udává jméno proměnné aktuálně zaměřeného editovatelného textového pole. Jestliže není žádné textové pole aktuálně zaměřeno, metoda udá null.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující kód udává jméno proměnné:

```
_root.jakykolivMovieKlip.mojeTextovePole.
```

Selection.setFocus

Syntaxe

```
Selection.setFocus(proměnná);
```

Argumenty

proměnná Řetězec udávající jméno proměnné spojené s textovým polem používajícím dot (tečkovou) nebo slash (lomítkovou) notaci.

Popis

Metoda; zaměřuje editovatelné textové pole spojené s určitou proměnnou.

Přehrávač

Flash 5 nebo novější.

Selection.setSelection

Syntaxe

```
Selection.setSelection(start,end);
```

Argumenty

start Počáteční index rozsahu výběru.

end Konečný index rozsahu výběru.

Popis

Metoda; nastavuje rozsah výběru aktuálně zaměřeného textového pole. Nový rozsah výběru bude začínat na indexu určeném v argumentu **start** a končit na indexu určeném v argumentu **end**. Indexy rozsahu výběru mají základ nula (kde první pozice je 0, druhá pozice je 1, atd.). Tato metoda nemá žádný efekt, jestliže není aktuálně zaměřeno žádné textové pole.

Přehrávač

Flash 5 nebo novější.

set

Syntaxe

proměnná = výraz;
`set`(proměnná, výraz);

Argumenty

proměnná Jméno schránky, která obsahuje hodnotu argumentu výraz.

výraz Hodnota (nebo fráze, která může být ohodnocena na hodnotu), která je připojená k proměnné.

Popis

Akce; připojuje hodnotu proměnné. Proměnná je schránka, která obsahuje nějakou informaci. Schránka samotná je vždy stejná, ale její obsah se může měnit. Při přehrávání animací, můžete změnou hodnoty proměnné nahrát a uložit informaci o tom, co uživatel udělal, nahrát hodnoty, které se mění, nebo ohodnotit, zda je podmínka **`true`**, nebo **`false`**.

Proměnné mohou obsahovat jak čísla, tak řetězce znaků. Každá animace a movie klip má své vlastní nastavení proměnných a každá proměnná má svoji vlastní hodnotu nezávislou na proměnných v jiných animacích nebo movie klipech.
ActionScript je netypizovaný jazyk. To znamená, že u proměnné nemusí být explicitně definováno, zda obsahuje číslo nebo řetězec. Flash zásadně interpretuje typ dat jako celé číslo nebo řetězec.

Pro zadávání nebo získání hodnot použijte příkaz **`set`** ve spojení s akcí **`call`**.

Přehrávač

Flash 4 nebo novější.

Příklad

Tento příklad nastavuje proměnnou nazvanou **`orig_x_pos`**, která ukládá původní pozici souřadnice **`x`** movie klipu **`ship`**, aby dále v animaci resetoval ship do startovního umístění:

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

Toto je ekvivalentní následujícího:

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

Viz také

`var`
`call`

setProperty

- **Syntaxe**

  `setProperty(cíl,vlastnost,výraz);`

- **Argumenty**

  cíl Cesta ke jménu instance movie klipu, jehož vlastnost je nastavována.

  vlastnost Vlastnost, která má být nastavena.

  výraz Hodnota, ke které je vlastnost nastavena.

- **Popis**

  Akce; mění během přehrávání animce vlastnost movie klipu.

- **Přehrávač**

  Flash 4 nebo novější.

- **Příklad**

  Tato zpráva nastavuje vlastnost `_alpha` movie klipu jménem `star` na 30 procent, když je stisknuto tlačítko:

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

- **Viz také**

`getProperty`

Sound (object)

Objekt Sound (Zvuk) umožňuje nastavit a kontrolovat zvuky v konkrétní instanci movie klipu, nebo na globální Časové ose, jestliže při vytváření nového objektu Sound nespecifikujete **cíl**. Pro vytvoření instance objektu Sound před tím, než voláte metody objektu Sound, musíte použít konstruktor `new Sound`.

Objekt Sound je podporován pouze pro Flash 5 Přehrávač.

Přehled metod objektu Sound

Metoda	Popis
`attachSound`	Připojuje zvuk specifikovaný v argumentu.
`getPan`	Udává hodnotu předchozího volání `setPan`.
`getTransform`	Udává hodnotu předchozího volání `setTransform`.
`getVolume`	Udává hodnotu předchozího volání `setVolume`.
`setPan`	Nastavuje pravé/levé vyrovnání zvuku.
`setTransform`	Nastavuje transformaci pro zvuk.
`setVolume`	Nastavuje úroveň hlasitosti pro zvuk.
`start`	Začíná hrát zvuk od začátku, nebo volitelně z vyrovnávacího bodu nastaveného v argumentu.
`stop`	Zastavuje specifikovaný hlas nebo všechny zvuky, které aktuálně hrají.

Konstruktor pro objekt Sound

- **Syntaxe**

```
new Sound();
new Sound(cíl);
```

- **Argumenty**

  cíl Instance movie klipu, ke které se připojuje objekt Sound. Tento argument je volitelný.

- **Popis**

  metoda; vytváří nový objekt Sound pro specifikovaný movie klip. Jestliže nespecifikujete **cíl**, objekt Sound kontroluje všechny zvuky v globální Časové ose.

- **Přehrávač**

  Flash 5 nebo novější.

- **Příklad**

```
GlobalniZvuk = new Sound();
MovieZvuk = new Sound(mojemovie);
```

Sound.attachSound

- **Syntaxe**
  mujZvuk.attachSound(„idJméno“);

- **Argumenty**
  idJméno Jméno pro novou instanci zvuku. Toto jméno je stejné jako jméno vložené pro identifikátor v dialog boxu Symbol **Linkage Properties**. Tento argument musí být vložen do citačních znamének " ".

- **Popis**
  Metoda; připojuje zvuk specifikovaný v argumentu **idJméno** ke specifikovanému objektu Sound. Zvuk musí být v knihovně aktuální animace a také musí být specifikovaný pro export v dialog boxu Symbol Linkage Properties. Pro spuštění zvuku volejte **Sound.start**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  Sound.start

Sound.getPan

- **Syntaxe**
  mujZvuk.getPan();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává úroveň pan nastavenou v posledním volání **setPan** jako celé číslo od -100 do 100. Nastavení pan kontroluje levo-pravou rovnováhu aktuálních a budoucích zvuků v animaci.
  Tato metoda je kumulativní s metodami **setVolume** nebo **setTransform**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  Sound.setPan
  Sound.setTransform

Sound.getTransform

- **Syntaxe**
  mujZvuk.getTransform();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává informaci o transformaci zvuku pro specifikovaný objekt Sound nastavenou v posledním volání **setTransform**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  Sound.setTransform

Sound.getVolume

- **Syntaxe**
  mujZvuk.getVolume();

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává úroveň hlasitosti zvuku jako celé číslo od 0 do 100, kde nula je vypnutý zvuk a 100 je plná hlasitost. Dané nastavení je 100.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  Sound.setVolume

Sound.setPan

Syntaxe

mujZvuk.setPan(pan);

Argumenty

pan Celé číslo určující pravo-levou rovnováhu pro zvuk. Rozsah platných hodnot je od -100 do 100, kde -100 používá pouze levý kanál, 100 používá pouze pravý kanál a 0 vyrovnává zvuk shodně mezi dvěma kanály.

Popis

Metoda; určuje, jak má zvuk hrát v pravém a levém kanálu (mluvítkách). Pro mono zvuky, ovlivňuje **pan**, přes které mluvítko (levé nebo pravé) bude zvuk hrán.

Tato metoda je kumulativní s metodami **setVolume** a **setTransform** a volání této metody ruší a updatuje předchozí nastavení **setPan** a **setTransformovat**.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá **setVolume** a **setPan** pro kontrolu objektu zvuk se specifikovaným cílem „u2“:

```
onClipEvent(mouseDown){
//vytvořit objekt zvuk
s = new Sound(this);
//připojit zvuk v knihovně
s.attachSound(„u2“);
//nastavit Hlasitost na 50%
s.setVolume(50);
//vypnout zvuk v pravém kanálu
s.setPan(-100);
//začít 30 sekund ve zvuku a hrát ho 5 krát
s.start(30, 5);
```

Viz také

Sound.setTransform
Sound.setVolume

Sound.setTransform

Syntaxe

mujZvuk.setTransform(soundTransformObject);

Argumenty

soundTransformObject Objekt vytvořený konstruktorem pro generický objekt Object.

Popis

Metoda; nastavuje informaci o transformaci zvuku pro objekt Sound. Tato metoda je kumulativní s metodami **setVolume** a **setPan** a volání této metody ruší a updatuje jakákoliv předchozí nastavení **setPan** a **setVolume**. Toto volání je pro expert uživatele, kteří chtějí ke zvukům přidat zajímavé efekty.

Zvuky používají značnou část prostoru disku a paměti. Protože stereo zvuky používají dvakrát více dat než mono zvuky, je všeobecně nejlepší použít 22-Khz 6-bit mono zvuky. Metodu **setTransform** můžete použít, pro hraní mono zvuků jako stereo, hraní stereo zvuků jako mono a pro připojení zajímavých efektů ke zvukům.

Argument **soundTransformObject** je objekt, který vytváříte pomocí použití konstrukční metody generického objektu Object s parametry určujícími, jak je zvuk rozdělován do pravého a levého kanálu (mluvítek).

Parametry pro **soundTransformObject** jsou následující:

ll Procentuální hodnota určující, kolik levého vstupu se má hrát v levém mluvítku (-100 až 100).

lr Procentuální hodnota určující, kolik pravého vstupu se má hrát v levém mluvítku (-100 až 100).

rr Procentuální hodnota určující, kolik pravého vstupu se má hrát v pravém mluvítku (-100 až 100).

rl Procentuální hodnota určující, kolik levého vstupu se má hrát v pravém mluvítku (-100 až 100).

Výsledek parametrů je reprezentován následujícím vzorcem:
levýVýstup = levý vstup *** ll +** pravý vstup *** lr**

pravýVýstup = pravý vstup *** rr +** levý vstup *** rl**

Hodnoty pro levý nebo pravý vstup jsou určeny typem (stereo nebo mono) zvuku v animaci.

Stereo zvuky rozdělují zvukový vstup shodně mezi levé a pravé mluvítko a mají následující dané nastavení transformace:

```
ll = 100
lr = 0
rr = 100
rl = 0
```

Mono zvuky hrají všechny zvukové inputy v levém mluvítku a mají následující dané nastavení transformace:

```
ll = 100
lr = 100
rr = 0
rl = 0
```

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad vytváří objekt zvukové transformace, který hraje levý i pravý kanál v levém kanálu:

```
mySoundTransformObject = new Object
mySoundTransformObject.ll = 100
mySoundTransformObject.lr = 100
mySoundTransformObject.rr = 0
mySoundTransformObject.rl = 0
```

Abyste aplikovali objektové zvukové transformace na objekt Sound, musíte vložit objekt do objektu Sound použitím **setTransform**:

```
mujZvuk.setTransform(mySoundTransformObject);
```

Následující jsou příklady nastavení, které mohou být nastaveny použitím **setTransform**, ale nemohou být nastaveny použitím **setVolume** nebo **setPan**, i kdyby byly kombinovány.

Tento kód hraje levý i pravý kanál skrz levý kanál:

```
mujZvuk.setTransform (soundTransformObjectLeft);
```


V kódu nahoře má sound**TransformObjectLeft** následující parametry:

```
ll = 100
lr = 100
rr = 0
rl = 0
```

Tento kód hraje stereo zvuk jako mono:

```
setTransform (soundTransformObjectMono);
```

V kódu nahoře má sound**TransformObjectMono** následující parametry:

```
ll = 50
lr = 50
rr = 50
rl = 50
```

Tento kód hraje levý kanál s poloviční kapacitou a přidává zbytek levého kanálu do pravého kanálu:

```
setTransform (soundTransformObjectHalf);
```

V kódu nahoře má soundTransformObjectHalf následující parametry:

```
ll = 50
lr = 0
rr = 100
rl = 50
```

Viz také

Konstruktor pro objekt Object

Sound.setVolume

Syntaxe

mujZvuk.`setVolume`(hlasitost);

Argumenty

hlasitost Číslo od 0 do 100 reprezentující úroveň hlasitosti. 100 je plná hlasitost a 0 je žádná hlasitost. Dané nastavení je 100.

Popis

Metoda; nastavuje hlasitost pro objekt Sound.

Tato metoda je kumulativní s metodami **`setPan`** a **`setTransform`**.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad nastavuje hlasitost na 50% a transferuje zvuk přes čas z levého mluvítka do pravého:

```
onClipEvent (load){
       i = -100;
       s = new Sound();
       s.setVolume(50);
}
onClipEvent (enterFrame){
       S.setPan(i++);
}
```

Viz také

`Sound.setPan`
`Sound.setTransform`

Sound.start

Syntaxe

mujZvuk.`start();`
mujZvuk.`start(`[secondOffset, loop]`);`

secondOffset Volitelný argument, který umožňuje začít hrát zvuk na určitém bodě. Například, jestliže máte 30sekundový zvuk a chcete, aby zvuk začal hrát uprostřed, určete 15 sekund pro argument **secondOffset**. Zvuk není opožděn o 15 sekund, ale začíná hrát na značce 15 sekund.

loop Volitelný argument, který umožňuje určit, kolikrát by měl zvuk smyčkovat (počet opakování-smyček).

Popis

Metoda; začíná hrát poslední připojený zvuk od začátku, jestliže není specifikován žádný argument, nebo začíná na bodě ve zvuku, který je určen argumentem **secondOffset.**

Přehrávač

Flash 5 nebo novější.

Viz také

`Sound.setPan`
`Sound.stop`

Sound.stop

Syntaxe

```
mujZvuk.stop();
mujZvuk.stop([„idJméno“]);
```

Argumenty

idJméno Volitelný argument určující, aby přestal hrát určitý zvuk. Argument **idJméno** musí být v citačních znaménkách (“ “).

Popis

Metoda; zastavuje všechny zvuky aktuálně hrající, jestliže není specifikován žádný argument, nebo pouze zvuk určený v argumentu **idJméno**.

Přehrávač

Flash 5 nebo novější.

Viz také

`Sound.start`

_soundbuftime

Syntaxe

`_soundbuftime =` **celé číslo;**

Argumenty

celé číslo Počet sekund před tím, než animace začne téct.

Popis

Vlastnost (všeobecná); udává počet sekund, o které se tekoucí zvuk prebuffer (zadrží přenášená data ve vyrovnávací paměti). Nastavená hodnota je 5 sekund.

Přehrávač

Flash 4 nebo novější.

startDrag

Syntaxe

```
startDrag(target);
startDrag(target, [lock]);
startDrag(target [, lock [, left, top, right, bottom ]]);
```

Argumenty

target Cílová cesta movie klipu, který se má táhnout.

lock Booleánská hodnota určující, zda je tažitelný movie klip zamknutý na střed pozice myši (`true`) nebo zamknutý na bod, kam uživatel poprvé v movie klipu kliknul(false). Tento argument je volitelný.

left, top, right, bottom Hodnoty relativní k souřadnicím rodiče movie klipu, které určují konstrukci obdélníku pro movie klip. Tyto argumenty jsou volitelné.

Popis

Akce; činí movie klip cíl tažitelný během hraní animace. V jednom čase může být tažitelný pouze jeden movie klip. Při výkonu operace `startDrag` zůstává movie klip tažitelný, dokud není explicitně zastaven akcí `stopDrag`, nebo dokud není volána akce `startDrag` pro jiný movie klip.

Příklad

Pro vytvoření movie klipu, který mohou uživatelé umístit do jakéhokoliv umístění, připojte uvnitř animace akce `startDrag` a `stopDrag` k tlačítku následovně:

```
on(press){
        startDrag("",true);
}
on(release){
        stopDrag();
}
```

Viz také

`stopDrag`
`_droptarget`

stop

Syntaxe
```
stop;
```

Argumenty
Žádné.

Popis
Akce; zastavuje aktuálně hrající animaci. Nejběžnější použití této akce je pro ovládání movie klipu tlačítky.

Přehrávač
Flash 3 nebo novější.

stopAllSounds

Syntaxe
```
stopAllSounds();
```

Argumenty
Žádné.

Popis
Akce; zastavuje všechny zvuky aktuálně hrající v animaci, bez zastavení hrací hlavy. Zvuky nastavené pro tok, budou pokračovat v hraní při pohybu hrací hlavy přes snímky, ve kterých jsou umístěny.

Přehrávač
Flash 3 nebo novější.

Příklad
Následující kód by byl použit na tlačítko, na které když se klikne, zastaví všechny zvuky v animaci:
```
on(release){
        stopAllSounds();
}
```

Viz také
`Sound`

stopDrag

Syntaxe
```
stopDrag();
```

Argumenty
Žádné.

Popis
Akce; zastavuje aktuální operaci tažení.

Přehrávač
Flash 4 nebo novější.

Příklad
Tato zpráva zastavuje akci tažení na instanci mc, kde uživatel uvolní tlačítko myši:
```
on(press){
        startDrag("mc");
}
on(release){
        stopDrag();
}
```

Viz také
`startDrag`
`_droptarget`

String (function)

Syntaxe

```
String(výraz);
```

Argumenty

výraz Číslo, Booleánská, proměnná nebo objekt, který se má konvertovat na řetězec.

Popis

Funkce; udává řetězcovou reprezentaci specifikovaného argumentu následovně: Jestliže je **x** Booleovská, obdržený řetězec je **`true`** nebo **`false`**.

Jestliže je **x** číslo, obdržený řetězec je decimální reprezentace čísla.

Jestliže je **x** řetězec, obdržený řetězec je **x**.

Jestliže je **x** objekt, obdržená hodnota je řetězcová reprezentace objektu generovaného voláním řetězcové vlastnosti pro objekt nebo voláním **`object.toString`**, pokud ale žádná taková vlastnost neexistuje.

Jestliže je **x** movie klip, obdržená hodnota je cílová cesta movie klipu v závorkách (/).

Jestliže **x** není definováno, obdržená hodnota je prázdný řetězec.

Přehrávač

Flash 3 nebo novější.

Viz také

`Object.toString`
`Number.toString`
`String (object)`
`""(string delimiter)`

" " (string delimiter)

Syntaxe

```
„text"
```

Argumenty

text Jakýkoliv text.

Popis

Řetězcový delimitátor (string delimiter); když je použitý před a za řetězcem, citační znaménka indikují, že je řetězec písmenný --- ne proměnná, numerická hodnota nebo jiný ActionScript prvek.

Přehrávač

Flash 4 nebo novější.

Příklad

Tato zpráva používá citační znaménka pro indikaci, že řetězec „Ostrov Prince Edwarda" je písmenný řetězec a ne hodnota proměnné:

```
provincie = „Ostrov Prince Edwarda"
```

Viz také

`String (object)`
`String (function)`

String (object)

Objekt String (Řetězec) je obal pro řetězec původního typu dat, který umožňuje používat metody a vlastnosti objektu String k manipulaci s původními typy řetězcových hodnot. Použitím funkce `String()` můžete konvertovat hodnotu jakéhokoliv objektu na řetězec.

Všechny metody objektu String, kromě **`concat, fromCharCode, slice`** a **`substr`**, jsou generické. To znamená, že metody samotné volají **`this.toString`** před tím, než provedou své operace a tyto metody můžete používat s dalšími ne-Řetězcovými objekty.

Metody objektu String, můžete volat použitím konstrukční metody **new String**, nebo použitím řetězcové písmenné hodnoty. Jestliže specifikujete písmenný řetězec, ActionScript interpreter ho automaticky konvertuje na dočasný objekt String, volá metodu a potom dočasný objekt String odloží. S písmenným řetězcem můžete použít také vlastnost **`String.length.`**

Je důležité, abyste si nepletli písmenný řetězec s instancí objektu String. V následujícím příkladě první řádek kódu vytváří písmenný řetězec **s1** a druhý řádek kódu vytváří instanci objektu String **s2**.

```
s1 = "foo"
s2 = new String("foo")
```

Je doporučováno, abyste používali písmenné řetězce, pokud specificky nepotřebujete použít objekt String, protože objekty String mohou mít kontraintuitivní chování.

Přehled metod objektu String

Metoda	Popis
`charAt`	Udává číslo korespondující s umístěním znaku v řetězci.
`charCodeAt`	Udává hodnotu znaku na daném indexu jako 16-bitové celé číslo mezi 0 a 65535.
`concat`	Kombinuje text dvou řetězců a udává nový řetězec.
`fromCharCode`	Udává řetězec vytvořený ze znaků specifikovaných v argumentech.
`indexOf`	Hledá řetězec a udává index hodnoty specifikované v argumentech. Jestliže se hodnota objeví více než jednou, je udán index prvního výskytu. Jestliže není hodnota nalezena, je udáno -1.
`lastIndexOf`	Udává poslední výskyt podřetězce uvnitř řetězce, který se objeví před počáteční pozicí specifikovanou v argumentu nebo udá -1, jestliže není nalezen.
`slice`	Vytahuje část řetězce a udává nový řetězec.
`split`	Rozděluje objekt String na pole řetězců oddělením řetězců na subřetězce.
`substr`	Udává specifikovaný počet znaků v řetězci, začínající na umístění specifikovaném v argumentu.
`substring`	Udává znaky mezi dvěma indexy specifikovanými v argumentech do řetězce.
`toLowerCase`	Konvertuje řetězec na malá písmena a udává výsledek.
`toUpperCase`	Konvertuje řetězec na velká písmena a udává výsledek.

Přehled vlastností pro objekt String

Metoda	Popis
`length`	Udává délku řetězce.

Konstruktor pro objekt String

- **Syntaxe**

  ```
  new String(hodnota);
  ```

- **Argumenty**

  hodnota Původní hodnota nového objektu String.

- **Popis**

  Konstruktor; vytváří nový objekt String.

- **Přehrávač**

  Flash 5 nebo novější.

- **Viz také**

  ```
  String (function)
  " " (string delimiter)
  ```

String.charAt

- **Syntaxe**

  ```
  mujRetezec.charAt(index);
  ```

- **Argumenty**

  index Číslo znaku v řetězci, který má být udán.

- **Popis**

  Metoda; udává znak specifikovaný argumentem index. Index prvního znaku v řetězci je 0. Jestliže index není číslo od 0 do **`string.length`** -1, je udán prázdný řetězec.

- **Přehrávač**

  Flash 5 nebo novější.

String.charCodeAt

- **Syntaxe**
  mujRetezec.`charCodeAt`(index);

- **Argumenty**
  index Číslo znaku, pro který je získána hodnota.

- **Popis**
  Metoda; udává hodnotu znaku specifikovaného **indexem**. Obdržená hodnota je 16-bitové celé číslo od 0 do 65535.

  Tato metoda je podobná **`string.charAt`** až na to, že obdržená hodnota je pro znak na specifickém umístění, namísto řetězce obsahujícího znak.

- **Přehrávač**
  Flash 5 nebo novější.

String.concat

- **Syntaxe**
  mujRetezec.`concat`(hodnota1,...hodnotaN) *;*

- **Argumenty**
  hodnota1,...hodnotaN Nula nebo více hodnot, které mají být spojeny.

- **Popis**
  Metoda; kombinuje specifikované hodnoty a udává nový řetězec. Jestliže je to nezbytné, každý argument hodnota je konvertován na řetězec a připojen, podle pořadí na konec řetězce.

- **Přehrávač**
  Flash 5 nebo novější.

String.fromCharCode

- **Syntaxe**
  mujRetezec.`fromCharCode`(c1,c2,...cN) *;*

- **Argumenty**
  c1,c2,...cN Znaky, které mají být předělány na řetězce.

- **Popis**
  Metoda; udává řetězec vytvořený ze znaků specifikovaných v argumentech.

- **Přehrávač**
  Flash 5 nebo novější.

String.indexOf

- **Syntaxe**
  `mujRetezec.indexOf(hodnota);`
  `mujRetezec.indexOf(hodnota, start);`

- **Argumenty**
  hodnota Celé číslo nebo řetězec určující podřetězec, který má být nalezen uvnitř **mujRetezec**.

  start Celé číslo specifikující počáteční bod podřetězce. Tento argument je volitelný.

- **Popis**
  Metoda; hledá řetězec a udává pozici prvního výskytu specifikované hodnoty. Jestliže hodnota není nalezena, metoda udá -1.

- **Přehrávač**
  Flash 5 nebo novější.

String.lastIndexOf

Syntaxe

```
mujRetezec.lastIndexOf(substring);
mujRetezec.lastIndexOf(substring, start);
```

Argumenty

substring (podřetězec) Celé číslo nebo řetězec specifikující řetězec, který má být nalezen.

start Celé číslo určující počáteční bod uvnitř podřetězce. Tento argument je volitelný.

Popis

Metoda; hledá řetězec a udává index posledního výskytu **podřetězce** nalezeného uvnitř volaného řetězce. Jestliže **podřetězec** není nalezen, metoda udá -1.

Přehrávač

Flash 5 nebo novější.

String.length

Syntaxe

```
string.length
```

Argumenty

Žádné.

Popis

Vlastnost; udává počet znaků ve specifikovaném objektu String. Index posledního znaku pro jakýkoliv řetězec je x.délka-1.

Přehrávač

Flash 5 nebo novější.

String.slice

Syntaxe

```
mujRetezec.slice(start, end);
```

Argumenty

start Číslo určující index počátečního bodu pro výřez. Jestliže je start záporné číslo, počáteční bod je určen od konce řetězce, kde -1 je poslední znak.

end Číslo určující index posledního bodu pro výřez. Jestliže není **end** specifikován, výřez zahrnuje všechny znaky od startu do konce řetězce. Jestliže je **end** záporné číslo, konečný bod je určen od konce řetězce, kde -1 je poslední znak.

Popis

Metoda; vytahuje výřez nebo subřetězec specifikovaného objektu String; potom ho udává jako nový řetězec bez modifikace původního objektu String. Obdržený řetězec zahrnuje znak start a všechny znaky až do (ne včetně) znaku end.

Přehrávač

Flash 5 nebo novější.

String.split

Syntaxe

```
mujRetezec.split(delimiter);
```

Argumenty

delimiter Znak použitý pro delimitaci řetězce.

Popis

Metoda; rozděluje objekt **String** rozdělením řetězce tam, kde se objeví argument delimiter a udává subřetězce v poli. Jestliže není specifikován žádný delimitátor, obdržené pole obsahuje pouze jeden prvek - samotný řetězec. Jestliže je **delimitátor** prázdný řetězec, každý znak v objektu **String** se stává prvkem v poli.

Přehrávač

Flash 5 nebo novější.

String.substr

- **Syntaxe**
  mujRetezec.`substr`(start, length);

- **Argumenty**

  start — Celé číslo, které indikuje pozici prvního znaku v subřetězci, který je vytvářen. Jestliže je start záporné číslo, počáteční pozice je určena od konce řetězce, kde -1 je poslední znak.

  length — Počet znaků v subřetězci, který je vytvářen. Jestliže délka není specifikována, subřetězec zahrnuje všechny znaky od startu do konce řetězce.

- **Popis**
  Metoda; udává znaky v řetězci od indexu specifikovaného v argumentu **start** až s počtem znaků určeným v argumentu **length**.

- **Přehrávač**
  Flash 5 nebo novější.

Sring.substring

- **Syntaxe**
  mujRetezec.`substring`(from, to);

- **Argumenty**

  from (od) — Celé číslo, které indikuje pozici prvního znaku v subřetězci, který je vytvářen. Platné hodnoty jsou od 0 do **`string.length`** -1.

  to (do) — Celé číslo, které ja 1+index posledního znaku v subřetězci, který je vytvářen. Platné hodnoty jsou od 1 do **`string.length`**. Jestliže není specifikován argument do, konec subřetězce je konec řetězce. Jestliže se od rovná do, metoda udá prázdný řetězec. Jestliže je od větší než do, argumenty jsou automaticky vyměněny před tím, než se vykoná funkce.

- **Popis**
  Metoda; udává řetězec sestávají ze znaků mezi body specifikovanými argumenty **od** a **do**.

- **Přehrávač**
  Flash 5 nebo novější.

String.toLowerCase

- **Syntaxe**
  mujRetezec.`toLowerCase()`;

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává kopii pro objekt String, se všemi velkými znaky konvertovanými na písmena malá.

- **Přehrávač**
  Flash 5 nebo novější.

String.toUpperCase

- **Syntaxe**
  mujRetezec.`toUpperCase()`;

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; udává kopii objektu String, se všemi malými znaky konvertovanými na velká písmena.

- **Přehrávač**
  Flash 5 nebo novější.

substring

- **Syntaxe**
  `substring`**(řetězec, index, počet)**;

- **Argumenty**

řetězec	Řetězec, ze kterého se má vytáhnout nový řetězec.
index	Číslo prvního znaku, který se má vytáhnout.
počet	Počet znaků, které se mají zahrnout do vytaženého řetězce, nezahrnuje znak index.

- **Popis**
  Řetězcová funkce; vytahuje část řetězce.

- **Přehrávač**
  Flash 4 nebo novější. Tato funkce byla zavržena ve Flash 5.

- **Viz také**
  `String.substring`

_target

- **Syntaxe**
  instancejméno`._target`

- **Argumenty**

instancejméno	Jméno movie klip instance.

- **Popis**
  Vlastnost (pouze pro čtení); udává cílovou cestu instance movie klipu specifikované v argumentu instancejméno.

- **Přehrávač**
  Flash 4 nebo novější.

targetPath

- **Syntaxe**
  `targetPath`**(movieClipObject)**;

- **Argumenty**

movieClipObject	Odkaz (například `_root` nebo `_parent`) na movie klip, pro který je získávána cílová cesta.

- **Popis**
  Funkce; udává řetězec obsahující cílovou cestu **movieClipObject**. Cílová cesta je obdržena v dot (tečkové) notaci. Pro získání cílové cesty ve slash (lomítkové) notaci použijte vlastnost `_target`.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  Následující příklady jsou ekvivalentní. První příklad používá tečkovou notaci a druhý příklad používá slash notaci.

```
targetPath (Tabule.Blok[index*2+1]){
play();
}
```

  Je ekvivalentní k:

```
tellTarget („Tabule/Blok:"+(index*2+1)){
play();
}
```

- **Viz také**
  `eval`

tellTarget

Syntaxe

```
tellTarget(cíl){
příkaz;
}
```

Argumenty

cíl Cílová cesta řetězce určující Časovou osu, která bude kontrolována.

příkaz Instrukce aplikované k cílované Časové ose.

Popis

Akce; aplikuje instrukce specifikované v argumentu **příkaz**, na Časovou osu určenou v argumentu **cíl**. Akce **tellTarget** je užitečná pro navigační kontroly. Připojte **tellTarget** k tlačítkům, které zastavují nebo startují movie klipy kdekoliv na Scéně. Také můžete udělat to, aby movie klipy šly na určitý snímek v tomto klipu. Například můžete připojit **tellTarget** k tlačítkům, které zastavují nebo startují movie klipy na Scéně nebo pobídnout movie klipy, aby skočily na určitý snímek. Akce **tellTarget** je velmi podobná akci **with**, s tím rozdílem, že **with** má jako cíl movie klip nebo jiný objekt, a **tellTarget** vyžaduje cílovou cestu k movie klipu a nemůže kontrolovat objekty.

Přehrávač

Flash 3 nebo novější. Tato akce je ve Flash 5 zavržena; je doporučeno používat akci **with**.

Příklad

Tento příklad **tellTarget** kontroluje instanci movie klipu **mic** na hlavní Časové ose. Snímek 1 movie klipu je prázdný a má akci **stop**, aby nebyl viditelný na Scéně. Když je kliknuto na tlačítko s následující akcí, **tellTarget** určí, aby hrací hlava v movie klipu **mic** šla na snímek 2 a hrála animaci, která zde začíná.

```
on(release){
	telltarget(„mic"){
		gotoAndPlay(2);
	}
}
```

Viz také

with

this

Syntaxe

```
this
```

Argumenty

Žádné.

Popis

Klíčové slovo; odkazuje na objekt nebo instanci movie klipu. Klíčové slovo **this** má stejný účel a funkci v ActionScript jako v JavaScript, s dodatečnou funkčností. V ActionScript, když se skript vykoná, **this** odkazuje na instanci movie klipu, která sskript obsahuje. **This** obsahuje odkaz na objekt, který obsahuje vykonanou metodu pokud je použito s vyvoláním metody,.

Přehrávač

Flash 5 nebo novější.

Příklad

V následujícím příkladě klíčové slovo this odkazuje na objekt Circle:

```
function Circle(radius){
	this.radius = radius;
	this.area = math.PI * radius * radius;
}
```

V následujícím příkazu připojenému ke snímku, klíčové slovo **this** odkazuje na aktuální movie klip:

```
//nastavení vlastnosti alfa aktuálního movie klipu na 20.
this._alpha=20;
```

V následujícím příkazu uvnitř ovladače **onClipEvent**, klíčové slovo **this** odkazuje na aktuální movie klip:

//když se movie klip natáhne, je inicializována funkce **startDrag** pro aktuální movie klip.

```
onClipEvent (load){
	startDrag (this, true);
}
```

Viz také

new

toggleHighQuality

Syntaxe

```
toggleHighQuality();
```

Argumenty

Žádné.

Popis

Akce; zapíná a vypíná antialiasing ve Flash Přehrávači. Antialiasing vyhlazuje okraje objektů a zpomaluje playback. Akce **`toggleHighQuality`** ovlivňuje všechny animace ve Flash Přehrávači.

Přehrávač

Flash 2 nebo novější.

Příklad

Následující kód by mohl být aplikován na tlačítko, které, když se na něj klikne, zapne nebo vypne antialiasing:

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

Viz také

`_quality`
`_highquality`

_totalframes

Syntaxe

instancejméno.`_totalframes`

Argumenty

instancejméno Jméno movie klipu, který má být ohodnocen.

Popis

Vlastnost (pouze pro čtení); ohodnocuje movie klip specifikovaný v argumentu instancejméno a udává celkový počet snímků v animaci.

Přehrávač

Flash 4 nebo novější.

trace

Syntaxe

`trace(`**výraz**`);`

Argumenty

výraz Výraz pro ohodnocení. Když používáte v animaci texty, výsledky argumentu výraz jsou zobrazeny v okně Output.

Popis

Akce; ohodnocuje výraz a zobrazuje výsledky do okna Output, v režimu test animace.

Akci **`trace`** použijte pro nahrání programovacích poznámek, nebo pro zobrazení zpráv k okně Output, pokud testujete animaci. Použijte parametr **výraz** pro kontrolu, zda podmínka existuje nebo pro zobrazení hodnot v okně Output. Akce **`trace`** ja podobná funkci **`alert`** v JavaScript.

Přehrávač

Flash 4 nebo novější.

Příklad

Tento příklad je ze hry, ve které musí být tažitelná instance movie klipu pojmenovaná **`zajic`** uvolněna na specifický cíl. Podmínkový výraz ohodnotí vlastnost **`_droptarget`** a vykoná různé akce v závislosti na tom, zda **`zajic`** je uvolněn. Akce **`trace`** je použita na konci skriptu pro ohodnocení umístění movie klipu **`zajic`** a zobrazí výsledky v okně Output. Jestliže se **`zajic`** nechová, tak, jak se očekávalo (například, jestliže se chytí na špatný cíl), hodnoty poslané do okna Output pomocí akce **`trace`** vám pomohou určit problém ve skriptu.

```
on(press){
zajic.startDrag();
}
on(release){
if(eval(_droptarget) != target){
	zajic._x = zajic_x;
	zajic._y = zajic_y;
}else {
zajic_x = zajic._x;
zajic_y = zajic._y;
target = "_root.pastvina";
}
trace("zajic_y = " + zajic_y);
trace("zajic_x = " + zajic_x);
stopDrag();
}
```

typeof

Syntaxe

```
typeof(výraz);
```

Argumenty

výraz Řetězec, movie klip, objekt nebo funkce.

Popis

Operátor; unární (jednočlenný) operátor umístěný před samostatným argumentem. Způsobuje, že Flash ohodnotí **výraz**; výsledkem je řetězec specifikující, zda je výraz řetězec, movie klip, objekt nebo funkce.

Přehrávač

Flash 5 nebo novější.

unescape

Syntaxe

```
unescape(x);
```

Argumenty

x Řetězec s hexadecimálními sekvencemi pro uniknutí.

Popis

Funkce nejvyšší úrovně; ohodnocuje argument **x** jako řetězec, dekóduje řetězec z URL-zakódovaného formátu (konvertuje všechny hexadecimální sekvence do ASCII znaků), a udává řetězec.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad ilustruje proces konverze **escape** -na- **unescape**.

```
escape(„Hello{[World]}“);
```

Výsledek je následující:

```
(„Hello%7B%5BWorld%5D%7D“);
```

Použijte unescape pro vrácení do původního formátu:

```
unescape („Hello%7B%5BWorld%5D%7D“)
```

Výsledek je následující:

```
Hello {[World]}
```


unloadMovie

Syntaxe

```
unloadMovie(umístění);
```

Argumenty

umístění Úroveň hloubky nebo cílový movie klip, ze kterého se má animace stáhnout.

Popis

Akce; odstraňuje animaci z Flash Přehrávače, která bylo předtím natažena použitím akce **loadMovie**.

Přehrávač

Flash 3 nebo novější.

Příklad

Následující příklad stahuje hlavní animaci a zanechává Scénu prázdnou:

```
unloadMovie (_root);
```

Následující příklad stahuje animaci na úrovni 15, když uživatel klikne na myš:

```
on(press){
      unloadMovie (_level15);
}
```

Viz také

loadMovie

updateAfterEvent

Syntaxe

`updateAfterEvent`**(movie klip událost)**;

Argumenty

movie klip událost	Můžete specifikovat jednu z následujících hodnot jako movie klip událost:
mouseMove	Akce je inicializována při každém pohybu myši. Použijte vlastnosti **`_xmouse`** a **`_ymouse`** k určení aktuální pozice myši.
mouseDown	Akce je inicializována, jestliže je stisknuto levé tlačítko myši.
mouseUp	Akce je inicializována, jestliže je levé tlačítko myši uvolněno.
keyDown	Akce je inicializována, jestliže je stisknuta klávesa. Použijte metodu **`Key.getCode`** pro získání informace o naposledy stisknuté klávese.
keyUp	Akce je inicializována, jestliže je klávesa uvolněna. Použijte metodu **`Key.getCode`** pro získání informace o naposledy stisknuté klávese.

Popis

Akce; updatuje display (nezávisle na nastavení snímků za sekundu v animaci) po dokončení klip události specifikované v argumentech. Tato akce není v seznamu panelu Flash Akce. Použití **`updateAfterEvent`** s akcemi tažení (drag), které specifikují vlastnosti **`_x`** a **`_y`** během pohybu myši, vám umožňuje přetahovat objekty hladce, bez efektu třepotání obrazovky.

Přehrávač

Flash 5 nebo novější.

Viz také

`onClipEvent`

_url

Syntaxe

instancejméno.`_url`

Argumenty

instancejméno	Cílový movie klip.

Popis

Vlastnost (pouze pro čtení); získává URL SWF souboru, ze kterého byl stažen movie klip.

Přehrávač

Flash 4 nebo novější.

var

Syntaxe

var proměnnáJméno1 [= hodnota1] [...,proměnnáJménoN [= hodnotaN]];

Argumenty

proměnnáJméno	Jméno proměnné, která se má deklarovat.
hodnota	Hodnota přiřazená k proměnné.

Popis

Akce; používá se pro deklaraci lokálních proměnných. Jestliže deklarujete lokální proměnné uvnitř funkce, proměnné jsou definovány pro funkci a zanikají na konci volání funkce. Jestliže proměnné nejsou deklarovány uvnitř bloku, ale seznam akce byl vykonán pomocí akce **`call`**, proměnné jsou lokální a zanikají na konci aktuálního seznamu. Jestliže proměnné nejsou deklarovány uvnitř bloku a aktuální seznam akce nebyl vykonán pomocí akce **`call`**, proměnné nejsou lokální.

Přehrávač

Flash 5 nebo novější.

_visible

Syntaxe

```
instancejméno._visible
instancejméno._visible = Booleovská;
```

Argumenty

Booleovská Vložte hodnotu **true** nebo **false**, abyste určili, zda je movie klip viditelný.

Popis

Vlastnost; určuje, zda je nebo není animace specifikovaná argumentem **instancejméno** viditelné. Movie klipy, které nejsou viditelné (vlastnost nastavená na **false**) jsou nedosažitelné. Například na tlačítko v movie klipu s vlastností **_visible** nastavenou na **false**, nemůže být kliknuto.

Přehrávač

Flash 4 nebo novější.

void

Syntaxe

```
void(výraz);
```

Argumenty

výraz Výraz jakékoliv hodnoty.

Popis

Operátor; unární operátor, který vyřazuje hodnotu **výrazu** a udává nedefinovanou hodnotu. Operátor **void** je často používán pro ohodnocení URL, aby se testovaly efekty stran bez zobrazení ohodnoceného výrazu v okně browseru. Operátor **void** je také používán ve srovnáních za použití operátoru **==** pro testování nedefinovaných hodnot.

Přehrávač

Flash 5 nebo novější.

while

Syntaxe

```
while(podmínka){
příkaz(y);
}
```


Argumenty

podmínka Příkaz, který je ohodnocen pokaždé, když je vykonána akce **while**. Jestliže se příkaz ohodnotí na **true**, běží výraz v příkazu.

příkaz(y) Výraz, který má běžet, jestliže je podmínka ohodnocena na **true**.

Popis

Akce; běží příkaz nebo série příkazů opakovaných ve smyčce, pokud je argument podmínka **true**. Na konci každé akce **while** Flash restartuje smyčku opakovaným testováním podmínky. Jestliže je podmínka **false** nebo rovna 0, Flash skočí na první příkaz za akcí **while**.

Smyčkování je běžně používáno pro vykonání akce, zatímco je počítadlo proměnné menší než určená hodnota. Na konci každé smyčky je počítadlo zvýšeno, dokud není dosaženo prahové hodnoty, podmínka už není true a smyčka končí.

Přehrávač

Flash 4 nebo novější.

Příklad

Tento příklad duplikuje pět movie klipů na Scéně, každý s náhodně generovanou **x** a **y** pozicí, **xscale** a **yscale** a vlastností **_alpha** pro dosažení efektu rozptýlení. Proměnná **foo** je inicializována s hodnotou 0. Argument **podmínka** je nastaven tak, že smyčka **while** poběží pětkrát nebo dokud je hodnota proměnné **foo** menší než 5. Uvnitř smyčky **while** je duplikován movie klip a **setProperty** je použito pro úpravu různých vlastností duplikovaného movie klipu. Poslední příkaz smyčky zvyšuje **foo** tak, že když hodnota dosáhne 5, argument **podmínka** se ohodnotí na **false** a smyčka nebude vykonána.

```
on(release){
        foo = 0;
        while(foo < 5){
                duplicateMovieClip("/flower", "mc"+foo, foo);
                setProperty("mc"+foo, _x, random(275));
                setProperty("mc"+foo, _y, random(275));
                setProperty("mc"+foo, _alpha, random(275));
                setProperty("mc"+foo, _xscale, random(200));
        setProperty("mc"+foo, _yscale, random(200));
        foo = foo + 1;
        }
}
```

Viz také

do…while
continue

_width

Syntaxe

```
instancejméno._width
instancejméno._width = hodnota;
```

Argumenty

hodnota Šířka movie v pixelech.

instancejméno Jméno instance movie klipu, pro který má být nastavena nebo získána vlastnost **_width**.

Popis

Vlastnost; nastavuje šířku animaci. V předchozích verzích Flash byly vlastnosti **height** a **_width** pouze pro čtení; ve Flash 5 mohou být nastaveny, stejně tak jako získány.

Přehrávač

Flash 4 jako vlastnost pouze pro čtení. Ve Flash 5 nebo novějším, může být tato vlastnost nastavena stejně tak jako získána.

Příklad

Následující příklad kódu nastavuje vlastnosti výška a šířka movie klipu, když uživatel klikne na myš:

```
onClipEvent(mouseDown){
        _width=200;
        _height=200;
}
```

Viz také

_height

with

Syntaxe

```
with (objekt){
příkaz(y);
}
```

Argumenty

objekt Instance ActionScript objektu nebo movie klip.

příkaz(y) Akce nebo skupina akcí vložených do složených závorek.

Popis

Akce; dočasně mění rozsah (nebo dílovou cestu) použitý pro ohodnocení výrazů a akcí v **příkazu(ech)**. Poté, co se akce **with** vykoná, rozsah řetězce je obnoven do původního stavu.

Objekt se stává kontextem, ve kterém jsou čteny vlastnosti, proměnné a funkce. Například, jestliže **objektem** je **myArray** a dvě z určených vlastností jsou **length** a **concat**, jsou tyto vlastnosti automaticky čteny jako **myArray.length** a **myArray.concat**. V jiném příkladě, jestliže je objekt **stat.california**, je to jako by byly volány jakékoliv akce nebo příkazy uvnitř akce **with**, zevnitř instance **california**.

Pro nalezení hodnoty identifikátoru v **příkazu(ech)** začíná ActionScript na začátku rozsahu řetězce specifikovaného **objektem** a hledá identifikátor na každé úrovni rozsahu řetězce v určitém pořadí.

Rozsah řetězce je použit akcí **with** pro vyřešení toho, aby identifikátory začínali první položkou v následujícím seznamu a pokračovali k poslední následovně:

- objekt odkazovaný nejvíce vnitřní akcí **with**
- objekt odkazovaný nejvíce vnější akcí **with**
- Aktivační objekt (Dočasný objekt, který je automaticky vytvořen při volání funkce, která obsahuje lokální proměnné volané ve funkci.)
- Movie klip obsahující aktuálně vykonávaný scénář
- Globální objekt (předdefinované objekty jako **Math**, **String**)

Ve Flash 5 akce **with** nahrazuje zavrženou akci **tellTarget**. Je doporučeno používat **with** namísto **tellTarget**, protože je to standardní rozšíření ActionScriptu na ECMA-262 standard. Principiální rozdíl mezi akcemi **with** a **tellTarget** je ten, že **with** bere odkaz na movie klip nebo jiný objekt jako svůj argument, zatímco **tellTarget** má řetězec cílová cesta identifikující movie klip a nemůže být použita pro cílování objektů. Pro nastavení proměnné uvnitř akce with musí být proměnná deklarovaná mimo akci **with** nebo musíte vložit celou cestu do Časové osy na které má proměnná být. Jestliže nastavíte proměnnou v akci **with** bez jejího deklarování, akce **with** bude hledat hodnotu vzhledem k rozsahu řetězce. Jestliže proměnná ještě neexistuje, nová hodnota bude nastavena na Časové ose, ze které je akce **with** volána.

Příklad

Následující příklad nastavuje vlastnosti **x** a **y** instance **nejakyJinyMovieKlip** a potom instruuje **nejakyJinyMovieKlip**, aby šel na snímek 3 a zastavil se:

```
with(nejakyJinyMovieKlip){
	_x = 50;
	_y = 100;
	gotoAndStop(3);
}
```

Následující kousek kódu ukazuje, jak byste napsali předchozí kód bez použití akce **with**:

```
nejakyJinyMovieKlip._x = 50;
nejakyJinyMovieKlip._y = 100;
nejakyJinyMovieKlip. gotoAndStop (3);
```

Tento kód by také mohl být napsán pomocí použití akce **tellTarget**:

```
tellTarget („nejakyJinyMovieKlip"){
	_x = 50;
	_y = 100;
	gotoAndStop (3);
}
```


Akce **with** je užitečná pro simultánní dosažení několika položek v seznamu rozsahu řetězce. V následující příkladu je zabudovaný objektu **Math** umístěn před rozsah řetězce. Nastavení **Math** jako daný objekt vyřeší identifikátory **cos, sin** a **PI** na **Math.cos**, **Math.sin** a **Math.PI**. Identifikátory **a, x, y** a **r** nejsou metody nebo vlastnosti objektu **Math**, ale protože existují v aktivačním rozsahu objektu funkce **polar**, vyřeší se ke koresponduj ícím lokálním proměnných.

```
function polar(r){
	var a,x,y
	with(Math){
	a = PI * r * r
	x = r * cos(PI)
	y = r * sin(PI/2)
}
trace("oblast = " + a)
trace("x = " + x)
trace("y = " + y)
}
```

Uhnízděné akce **with** můžete použít pro dosažení informace v několika prostorech. V následujícím příkladu instance **fresno** a instance **salinas** jsou děti instance **california**. Zpráva nastavuje hodnoty **_alpha** instancí **fresno** a salinas bez změny hodnoty **_alpha** instance **california**.

```
with (california){
	with (fresno){
		_alfa = 20;
	}
	with (salinas){
		_alfa = 40;
	}
}
```

Viz také

tellTarget

_x

Syntaxe

instancejméno._x
instancejméno._x = celé číslo

Argumenty

celé číslo	Lokální souřadnice x movie.
instancejméno	Jméno movie klip instance.

Popis

Vlastnost; nastavuje souřadnici **x** movie relativně k lokálním souřadnicím rodičovského movie klipu. Jestliže je movie klip na hlavní Časové ose, potom jeho systém souřadnic odpovídá hornímu levému rohu Scény jako (0,0). Jestliže je movie klip uvnitř jiného movie klipu, který má transformace, movie klip je v lokálním systému souřadnic vloženého movie klipu. Tedy, pro movie klip otočený 90° proti směru hodinových ručiček, zdědí dítě movie klipu systém souřadnic, který je otočen o 90° proti směru hodinových ručiček. Souřadnice movie klipu odpovídají pozici registračního bodu.

Přehrávač

Flash 3 nebo novější.

Viz také

_y
_xscale

XML (object)

Metody a vlastnosti XML objektu použijte pro natažení, analýzu, poslání, vytvoření a manipulaci se stromy XML dokumentu.

Musíte použít konstruktor **new XML()** pro vytvoření instance XML objektu před tím, než voláte jakékoliv metody XML objektu.

XML je podporován Flash 5 a pozdějšími verzemi Flash Přehrávače.

Přehled metod objektu XML

Metoda	Popis
appendChild	Připojuje node na konec specifikovaného seznamu dítěte objektu.
cloneNode	Klonuje specifikovaný node a volitelně, rekurzivně klonuje všechny děti.
createElement	Vytváří nový XML prvek.
createTextNode	Vytváří nový XML textový node.
hasChildNodes	Udává true, jestliže specifikovaný node má dětské nody; jinak udává false.
insertBefore	Vkládá node před existující node ve specifikovaném seznamu dítěte node.
load	Natahuje dokument (specifikovaný XML objektem) z URL.
onLoad	Funkce zpětného volání pro load a sendAndLoad.
parseXML	Analyzuje XML dokument ve specifikovaném stromu XML objektu.
removeNode	Odstraní specifikovaný node z jeho rodiče.
send	Posílá specifikovaný XML objekt do URL.
sendAndLoad	Posílá specifikovaný XML objekt do URL a natahuje odpověď serveru do jiného XML objektu.
toString	Konvertuje specifikovaný node a jakékoliv dítě na XML text.

Přehled vlastností pro XML objekt

Metoda	Popis
`docTypeDecl`	Nastavuje a udává informaci o DOCTYPE deklaraci XML dokumentu.
`firstChild`	Odkazuje první dítě v seznamu na specifikovaný node.
`lastChild`	Odakzuje poslední díte v seznamu na specifikovaný node.
`loaded`	Kontroluje, zda byl specifikovaný XML objekt natažen.
`nextSibling`	Odkazuje na dalšího sourozence v rodičovském seznamu dítěte node.
`nodeName`	Udává jméno tagu XML prvku.
`nodeType`	Udává typ specifikovaného node (XML prvek nebo textový node).
`nodeValue`	Udává text specifikovaného node, jestliže je node textový node.
`parentNode`	Udává rodičovský node specifikovaného node.
`previousSibling`	Udává předchozího sourozence v rodičovském seznamu dítěte node.
`status`	Udává numerický stav kódu udávající úspěch nebo neúspěch operace analyzování XML dokumentu.
`xmlDecl`	Nastavuje a udává informaci o dokument deklaraci XML dokumentu.

Vybraný přehled pro XML objekt

Metoda	Popis
`attributes`	Udává asociační pole obsahující všechny atributy specifikovaného node.
`childNodes`	Udává pole obsahující odkazy na dětské nody specifikovaného node.

Konstruktor pro XML objekt

- **Syntaxe**

```
new XML();
new XML(source);
```

- **Argumenty**

source XML dokument analyzovaný pro vytvoření nového XML objektu.

- **Popis**

Konstruktor; vytváří nový XML objekt. Pro vytvoření instance XML objektu před tím, než voláte jakoukoliv metodu XML objektu, musíte použít konstrukční metodu.

První syntaxe konstruuje nový, prázdný XML objekt.

Druhá syntaxe konstruuje nový XML objekt analýzou XML dokumentu specifikovaného v argumentu source a osídluje nově vytvořený XML objekt s výsledným stromem XML dokumentu.

Poznámka: Metody createTextNode a createTextNode jsou konstrukční metody pro vytvoření prvků a textových nodů ve stromu XML dokumentu.

- **Přehrávač**

Flash 5 nebo novější.

- **Příklad**

Následující příklad vytváří nový prázdný XML objekt:

```
myXML = new XML();
```

- **Viz také**

`XML.createTextNode`
`XML.createTextNode`

XML.appendChild

- **Syntaxe**

```
myXML.appendChild(childNode);
```

- **Argumenty**

childNode Dětský node, který má být přidán ke specifikovanému XML seznamu dítěte objektu.

- **Popis**

Metoda; připojuje specifikovaný dětský node k XML seznamu dítěte objektu. Připojený dětský node je umístěn ve stromové struktuře, když je odstraněn ze svého existujícího rodičovského node.

- **Přehrávač**

Flash 5 nebo novější.

- **Příklad**

Následující příklad klonuje poslední node z `doc1` a připojuje ho k `doc2`:

```
doc1 = new XML(src1);
doc2 = new XML();
node = doc1.lastChild.cloneNode(true);
doc2.appendChild(node);
```

XML.attributes

Syntaxe
```
myXML.attributes;
```

Argumenty
Žádné.

Popis
Výběr (číst-psát); udává asociativní pole obsahující všechny atributy specifikovaného XML objektu.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad píše jména XML atributů do okna Output:
```
str = "<mytag name=\"Val\"> polozka </mytag>";
doc = new XML(str);
y = doc.fristChild.attributes.name;
      trace (y);
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order
      trace(z);
```

Do okna Output je psáno následující:
```
Val
First
```


XML.childNodes

Syntaxe
```
myXML.childNodes;
```

Argumenty
Žádné.

Popis
Výběr (pouze pro čtení); udává pole specifikovaného XML dítěte objektu. Každý prvek v poli je odkaz na XML objekt, který reprezentuje dětský node. Toto je vlastnost pouze pro čtení a nemůže být použitý pro manipulaci s dětskými nodes. Použijte metody **`appendChild, insertBefore`** a **`removeNode`** pro manipulaci s dětskými nodes.

Tento výběr není definován pro textové nodes (nodeTyp==3).

Přehrávač
Flash 5 nebo novější.

XML.cloneNode

Syntaxe
myXML.`cloneNode`(deep);

Argumenty
deep Booleovská hodnota specifikující, zda jsou děti specifikovaného XML objektu rekurzivně klonovány.

Popis
Metoda; konstruuje a udává nový XML node se stejným typem, jménem, hodnotou a atributy jako má specifikovaný XML objekt. Jestliže je **deep** nastaveno na **`true`**, všechny dětské nody jsou rekurzivně klonovány, výsledkem je přesná kopie původního stromu dokumentu objektu.

Přehrávač
Flash 5 nebo novější.

XML.createElement

Syntaxe

myXML.createElement(jméno);

Argumenty

jméno Jméno tagu XML prvku, který je vytvářen.

Popis

Metoda; vytváří nový XML prvek se jménem specifikovaným v argumentu. Nový prvek nemá původně žádného rodiče ani žádné děti. Metoda udává odkaz na nově vytvořený XML objekt reprezentující prvek. Tato metoda a **createTextNode** jsou konstrukční metody pro vytváření nodů pro XML objekt.

Přehrávač

Flash 5 nebo novější.

XML.createTextNode

Syntaxe

myXML.createTextNode(text);

Argumenty

text Text použitý pro vytvoření nového textového node.

Popis

Metoda; vytváří nový XML textový node se specifikovaným textem. Nový node nemá původně žádného rodiče a textové nody nemohou mít děti. Tato metoda udává odkaz na XML objekt reprezentující nový textový node. Tato metoda a **createElement** jsou konstrukční metody pro vytváření nodů pro XML objekt.

Přehrávač

Flash 5 nebo novější.

XML.docTypeDecl

Syntaxe

myXML.docTypeDecl;

Argumenty

Žádné.

Popis

Vlastnost; nastavuje a udává informaci o XML DOCTYPE deklaraci dokumentu. Poté, co byl XML text analyzován na XML objekt, je vlastnost XML objektu XML.**docTypeDecl** nastavena na text XML DOCTYPE deklarace dokumentu. Například **<!DOCTYPE zdraví SYSTEM „ahoj.dtd">**. Tato vlastnost je sada použití řetězcové reprezentace DOCTYPE deklarace, ne XML node objektu.

XML analyzátor ActionScript není analyzátor potvrzující platnost. DOCTYPE deklarace je čtena analyzátorem a uložena ve vlastnosti **docTypeDecl**, ale není provedena DTD validace.

Jestliže nedošlo k setkání se žádnou DOCTYPE deklarací během operace analyzování, je **XML.docTypeDecl** nastaven na nedefinovaný. **XML.toString** vydává obsahy **XML.docTypeDecl** okamžitě poté, co je XML deklarace uložena v **XML.xmlDecl** a před jakýmkoliv dalším textem v XML objetu. Jestliže je **XML.docTypeDecl** nedefinovaný, výstupem je žádná DOCTYPE deklarace.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá XML.docTypeDecl pro nastavení DOCTYPE deklarace pro XML objekt.

```
mujXML.docTypeDecl = " <!DOCTYPE zdraví SYSTÉM \ "ahoj.dtd \"> ";
```

Viz také

XML.toString
XML.xmlDecl

XML.firstChild

Syntaxe
```
myXML.firstChild;
```

Argumenty
Žádné.

Popis
Vlastnost (pouze pro čtení); ohodnocuje specifikovaný XML objekt a odkazuje na první dítě v rodičovském seznamu dítěte node. Tato vlastnost je **null**, jestliže node nemá děti. Tato vlastnost je nedefinována, jestliže je node textový node. Toto je vlastnost pouze pro čtení a nemůže být použita pro manipulaci s dětskými nody; pro manipulaci s dětskými nody, použijte metody **appendChild, insertBefore** a **removeNode**.

Přehrávač
Flash 5 nebo novější.

Viz také
XML.appendChild
XMl.insertBefore
XML.removeNode

XML.hasChildNodes

Syntaxe
```
myXML.hasChildNodes();
```

Argumenty
Žádné.

Popis
Metoda; ohodnocuje specifikovaný XML objekt a udává true, jestliže tam jsou dětské nody; jinak udává false.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad používá informaci z XML objektu ve funkci definované uživatelem:

```
if (rootNode.hasChildNodes()){
myfunc (rootNode.firstChild);
}
```


XML.insertBefore

Syntaxe
```
myXML.insertBefore(childNode, beforeNode);
```

Argumenty

childNode	Node, který má být vložen.
beforeNode	Node před bodem vložení pro childNode.

Popis
Metoda; vkládá nový dětský node do XML seznamu dítěte objektu, před **beforeNode**.

Přehrávač
Flash 5 nebo novější.

XML.lastChild

Syntaxe
```
myXML.lastChild;
```

Argumenty
Žádné.

Popis
Vlastnost (pouze pro čtení); ohodnocuje XML objekt a odkazuje na poslední dítě v rodičovském seznamu dítěte node. Tento způsob udává **null**, jestliže node nemá děti. Toto je vlastnost pouze pro čtení a nemůže být použita pro manipulaci s dětskými nody; pro manipulaci s dětskými nody, použijte metody **appendChild, insertBefore** a **removeNode**.

Přehrávač
Flash 5 nebo novější.

Viz také
XML.appendChild
XMl.insertBefore
XML.removeNode

XML.load

Syntaxe
```
myXML.load(url);
```

Argumenty
url Url, kde je umístěn XML dokument, který se má natáhnout. URL musí být ve stejné subdoméně jako URL, kde movie aktuálně sídlí.

Popis
Metoda; natahuje XML dokument ze specifikovaného URL a nahrazuje obsahy specifikovaného XML objektu daty nataženého XML. Proces natahování je asynchronní; nekončí okamžitě poté, co je vykonána metoda load. Když je load vykonáno, vlastnost objektu loaded je nastavena na false. Když XML data dokončí natahování, vlastnost loaded je nastavena na **true** a je vyvolána metoda onLoad. XML data nejsou analyzována, dokud nejsou kompletně natažena. Jestliže XML objekt předtím obsahoval jakékoliv XML stromy, jsou vyřazeny.

Můžete specifikovat svou vlastní funkci zpětného volání místo metody **onLoad**.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující je jednoduchým příkladem použití **XML.load**:
```
doc = new XML();
doc.load(„theFile.xml");
```

Viz také
XML.onLoad
XML.loaded

XML.loaded

Syntaxe
```
myXML.loaded;
```

Argumenty
Žádné.

Popis
Vlastnost (pouze pro čtení); určuje, zda proces natahování dokumentu inicializovaný pomocí **XML.load** je dokončen. Jestliže se proces úspěšně dokončí, metoda udává **true**; jinak udává **false**.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující příklad používá **XML.loaded** v jednoduchém skriptu.
```
if (doc.loaded){
    gotoAndPlay(4)
}
```

XML.nextSibling

Syntaxe
```
myXML.nextSibling;
```

Argumenty
Žádné.

Popis
Vlastnost (pouze pro čtení); ohodnocuje XML objekt a odkazuje na dalšího sourozence v rodičovském seznamu dítěte node. Tato metoda udává **null**, jestliže node nemá další sourozenecký node. Toto je vlastnost pouze pro čtení a nemůže být použita pro manipulaci s dětskými nody. Pro manipulaci s dětskými nody, použijte metody **appendChild**, **insertBefore** a **removeNode**.

Přehrávač
Flash 5 nebo novější.

Viz také
XML.appendChild
XML.insertBefore
XML.removeNode

XML.nodeName

- **Syntaxe**
  myXML.nodeName;

- **Argumenty**

Žádné.

- **Popis**
  Vlastnost; bere nebo udává jméno node XML objektu. Jestliže je XML objekt XML prvek **(nodeType==1)**, je **nodeName** jméno tagu reprezentujícího node v XML souboru. Například **TITLE** je **nodeName** HTML tagu **TITLE**. Jestliže je XML objekt textový node **(nodeType==3)**, nodeName je **null.**

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  XML.nodeType

XML.nodeType

- **Syntaxe**
  myXML.nodeType;

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost (pouze pro čtení); bere nebo udává hodnotu **nodeType**, kde 1 je XML prvek a 3 je textový **node**.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  XML.nodeValue

XML.nodeValue

- **Syntaxe**
  myXML.nodeValue;

- **Argumenty**
  Žádné.

- **Popis**
  Vlastnost; udává hodnotu node XML objektu. Jestliže je XML objekt textový node, **nodeType** je 3 a **nodeValue** je text node. Jestliže je XML objekt XML prvek, má **nodeValue null** a je pouze pro čtení.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  XML.nodeType

XML.onLoad

- **Syntaxe**
  myXML.onLoad(success);

- **Argumenty**
  success Booleánská hodnota indikující, zda byl XML objekt úspěšně natažen pomocí operace XML.load nebo XML.sendAndLoad.

- **Popis**
  Metoda; vyvolán Flash Přehrávačem, když je XML dokument obdržen ze serveru. Jestliže je XML dokument obdržen úspěšně, argument success je **true**. Jestliže dokument nebyl obdržen nebo jestliže se v odkazu obdržení ze serveru objevila chyba, argument success je **false**. Nastavená implementace této metody není aktivní. Pro přepsání nastavené implementace musíte připojit funkci obsahující vaše vlastní akce.

- **Přehrávač**
  Flash 5 nebo novější.

Příklad

Následující příklad vytváří animaci - jednoduchou aplikaci výlohy e-obchodu. Používáme metodu **sendAndLoad** pro přenos XML prvku obsahujícího jméno a heslo uživatele a instalujeme ovladač onLoad pro ovládání odpovědi ze serveru.

```
var myLoginReply = new XML();
myLoginReply.onLoad = myOnLoad;
myXML.sendAndLoad("http://www.samplestore.com/login.cgi",
myLoginReply);
function myOnLoad(success){
        if (success){
          if (e.firstChild.nodeName =="LOGINREPLY"&&
                    e.firstChild.attributes.status =="OK"){
           gotoAndPlay("loggedIn")
          }else {
           gotoAndStop("loginFailed")
          }
       }else {
          gotoAndStop("connectionFailed")
       }
}
```

Viz také

function
XML.load
XML.sendAndLoad

XML.parentNode

Syntaxe

myXML.parentNode;

Argumenty

Žádné.

Popis

Vlastnost (pouze pro čtení); odkazuje na rodičovský node specifikovaného XML objektu nebo udává null, jestliže node nemá žádného rodiče. Toto je vlastnost pouze pro čtení a nemůže být použita pro manipulaci s dětskými nody; pro manipulaci s dětskými nody, použijte metody **appendChild, insertBefore** a **removeNode**.

Přehrávač

Flash 5 nebo novější.

XML.parseXML

Syntaxe

myXML.parseXML(source);

Argumenty

source	XML text, který má být analyzován a propuštěn do specifikovaného XML objektu.

Popis

Metoda; analyzuje XML text specifikovaný v argumentu source a osídluje specifikovaný XML objekt s výsledným stromem XML. Jakékoliv existující stromy v XML objektu jsou vyřazeny.

Přehrávač

Flash 5 nebo novější.

XML.previousSibling

Syntaxe

myXML.previousSibling;

Popis

Vlastnost (pouze pro čtení); ohodnocuje XML objekt a odkazuje na předchozího sourozence v rodičovském seznamu dítěte node. Udává **null**, jestliže node nemá předchozí sourozenecký Node. Toto je vlastnost pouze pro čtení a nemůže být použita pro manipulaci s dětskými nody; pro manipulaci s dětskými nody, použijte metody **appendChild, insertBefore** a **removeNode.**

Přehrávač

Flash 5 nebo novější.

XML.removeNode

Syntaxe

```
myXML.removeNode();
```

Argumenty

Žádné.

Popis

Metoda; odstraňuje specifikovaný XML objekt z jeho rodiče.

Přehrávač

Flash 5 nebo novější.

XML.send

Syntaxe

```
myXML.send(url);
myXML.send(url, window);
```

Argumenty

url Určená URL pro specifikovaný XML objekt.

window Okno browseru, ve kterém se mají zobrazit data získaná serverem:
`_self` - specifikuje aktuální rám v aktuálním okně
`_blank` - specifikuje nové okno
`_parent` - specifikuje rodiče aktuálního rámu
`_top` – specifikuje rám nejvyšší úrovně v aktuálním okně.

Popis

Metoda; kóduje specifikovaný XML objekt do XML dokumentu a posílá ho do specifikovaného URL pomocí použití způsobu POST.

Přehrávač

Flash 5 nebo novější.

XML.sendAndLoad

Syntaxe

```
myXML.sendAndLoad(url, targetXMLobject);
```

Argumenty

url Určená URL pro specifikovaný XML objekt. URL musí být ve stejné subdoméně jako URL, ze kterého je animace stahována.

targetXMLobject XML objekt vytvořený pomocí XML konstrukční metody, který obdrží informaci ze serveru.

Popis

Metoda; kóduje specifikovaný XML objekt do XML dokumentu, posílá ho do specifikovaného URL pomocí použití způsobu `POST`, stahuje odpověď serveru a potom ji natahuje do **cílXMLobjekt** specifikovaného v argumentech. Odpověď serveru je natažena stejným způsobem pomocí použití metody `load`.

Přehrávač

Flash 5 nebo novější.

Viz také

`XML.load`

XML.status

Syntaxe
myXML.`status;`

Argumenty
Žádné.

Popis
Vlastnost; automaticky nastavuje a udává numerickou hodnotu indikující, zda byl XML dokument úspěšně analyzován do XML objektu. Následující je seznam numerických kódů stavů a jejich popis:

- 0 Žádná chyba; analýza je úspěšně dokončena.
- -2 Část CDATA nebyla řádně ukončena.
- -3 XML deklarace nebyla řádně ukončena.
- -4 Deklarace DOCTYPE nebyla řádně dokončena.
- -5 Komentář nebyl řádně dokončen.
- -6 XML prvek byl malformed.
- -7 Nedostatek paměti.
- -8 Hodnota atributu nebyla řádně dokončena.
- -9 Start-tag neodpovídá koncovému štítku.
- -10 Došlo na setkání k koncovým tagem, bez toho, že by odpovídal start-tagu.

Přehrávač
Flash 5 nebo novější.

XML.toString

Syntaxe
myXML.`toString();`

Argumenty
Žádné.

Popis
Metoda; ohodnocuje specifikovaný XML objekt, konstruuje textovou reprezentaci XML struktury včetně node, dětí a atributů a udává výsledek jako řetězec.

Pro XML objekty nejvyšší úrovně (to jsou ty, které byly vytvořené konstruktorem) udává **`XML.toString`** deklaraci dokumentu XML (uloženou v **`XML.xmlDecl`**), následovanou DOCTYPE deklarací dokumentu (uloženou v **`XML.docTypeDecl`**), následovanou textovou reprezentací všech XML nodů v objektu. XML deklarace není výstupem, jestliže **`XML.xmlDecl`** je nedefinovaný. Deklarace DOCTYPE není výstupem, jestliže není **`XML.docTypeDecl`** definovaný.

Přehrávač
Flash 5 nebo novější.

Příklad
Následující kód je příkladem metody **`XML.toString`**:

```
node = new XML(„<h1>text</h1>„);
trace(node.toString());
posílá
<H1>test</H1>
do okna výstup
```

Viz také
`XML.xmlDecl`
`XML.docTypeDecl`

XML.xmlDecl

Syntaxe

```
myXML.xmlDecl;
```

Argumenty

Žádné.

Popis

Vlastnost; nastavuje a udává informaci o deklaraci XML dokumentu. Poté, co je XML dokument analyzován na XML objekt, jeho vlastnost je nastavena pomocí použití textu deklarace XML dokumentu. Tato vlastnost je nastavena pomocí použití řetězcové reprezentace XML deklarace, ne XML node objektu. Jestliže nedošlo k setkání se žádnou XML deklarací během operace analýzy, vlastnost je nastavena ne nedefinována. **`XML.toString`** udává obsah **`XML.xmlDecl`** před jakýmkoliv jiným textem v XML objektu. Jestliže **`XML.xmlDecl`** obsahuje typ **`undefined`** (nedefinováno), není výstupem žádná XML deklarace.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá **`XML.xmlDecl`** pro nastavení deklarace XML dokumentu pro XML objekt:

```
mujXML.xmlDecl="<?xml verze=\"1.0\"?>";
```

Viz také

`XML.toString`
`XML.docTypeDecl`

XMLSocket (object)

XMLSocket objekt implementuje klientské zásuvky, které umožňují počítačům, na kterých běží Flash Přehrávač, komunikovat se serverovým počítačem identifikovaným IP adresou nebo jménem domény.

Použití XMLSocket objektu

Pro použití XMLSocket objektu musí serverový počítač běžet daemon (program), který chápe protokol používaný XMLSocket objektem. Protokol je následující:

- XML zprávy jsou posílány přes plně-duplexní TCP/IP tekoucí zásuvkové spojení.
- Každá XML zpráva je kompletní XML dokument, ukončený nulovým bytem.
- Přes jedno XMLSocket spojení, může být poslán a obdržen neomezený počet XML zpráv.

XMLSocket objekt je užitečný pro aplikace klient-server, které požadují nízkou latentnost, jako jsou například chat systémy v reálném čase. Tradiční HTTP-založené chat řešení často odesílá server a stahuje nové zprávy použitím HTTP požadavku. Naproti tomu, XMLSocket chat řešení ovládá otevřené spojení se serverem, které serveru umožňuje okamžitě poslat přicházející zprávy bez požadavku od klienta.

Nastavení serveru pro komunikaci s XMLSocket objektem může být problematické. Jestliže vaše aplikace nepožaduje interaktivitu v reálném čase, použijte namísto XMLSocket objektu akci **`loadVariables`** nebo Flash HTTP-založené XML server spojení (**`XML.load, XML.sendAndLoad, XML.send`**),.

Pro použití metod XMLSocket objektu musíte pro vytvoření nového XMLSocket objektu nejprve použít konstruktor **`new XMLSocket,`**.

XMLSocket a bezpečnost

Protože XMLSocket objekt zavádí a podporuje otevřené spojení se serverem, byly z bezpečnostních důvodů zavedeny následující restrikce na XMLSocket objekt:

- Metoda XML`Socket.connect` může spojovat pouze s TCP port čísly vyššími nebo rovnými 1024. Jeden význam této restrikce je, že serverové daemons (programy), které komunikují s XMLSocket objektem musí být také připojené k port číslům vyšším nebo rovným 1024. Port čísla nižší než 1024 jsou často používány systémovými službami jako FTP, Telnet a HTTP, tedy zatarasujícími XMLSocket objekt z těchto portů. Restrikce na čísla portů limituje možnost, že tyto zdroje budou nepatřičně dosaženy a zneužity.

- Metoda XML`Socket.connect` může spojovat pouze s počítači ve stejné subdoméně, kde sídlí SWF soubor. Tato restrikce se neaplikuje na animaci běžící mimo lokální disk. (Tato restrikce je stejná jako bezpečnostní pravidla pro `loadVariables`, `XML.sendAndLoad` a `XML.load`.)

Přehled metod objektu XMLSocket

Metoda	Popis
`close`	Zavírá otevřené zásuvkové spojení.
`connect`	Zavádí spojení se specifikovaným serverem.
`onClose`	Funkce zpětného volání, která je vyvolána, když je zavřeno XMLSocket spojení.
`onConnect`	Funkce zpětného volání, která je vyvolána, když je zavedeno XMLSocket spojení.
`onXML`	Funkce zpětného volání, která je vyvolána, když přijde XML objekt ze serveru.
`send`	Posílá XML objekt do serveru.

Konstruktor pro XMLSocket objekt

- **Syntaxe**
  ```
  new XMLSocket();
  ```

- **Argumenty**
  Žádné.

- **Popis**
  Konstruktor; vytváří nový XML`Socket` objekt. XML`Socket` objekt není původně spojen se žádným serverem. Pro spojení objektu se serverem musíte volat metodu `XMLSocket.connect`.

- **Přehrávač**
  Flash 5 nebo novější.

- **Příklad**
  ```
  myXMLSocket = new XMLSocket();
  ```

- **Viz také**
  `XMLSocket.connect`

XMLSocket.close

- **Syntaxe**
  ```
  myXMLSocket.close();
  ```

- **Argumenty**
  Žádné.

- **Popis**
  Metoda; zavírá spojení specifikované XML`Socket` objektem.

- **Přehrávač**
  Flash 5 nebo novější.

- **Viz také**
  `XMLSocket.connect`

XMLSocket.connect

Syntaxe

myXMLSocket.`connect`(host, port);

Argumenty

host Plně kvalifikované jméno DNS domény nebo IP adresa ve formě **aaa.bbb.ccc.ddd**. Také můžete specifikovat **`null`** pro spojení s hostitelským serverem, na kterém sídlí movie.

port TCP port číslo na hostiteli, použité pro zavedení spojení. Port číslo musí být 1024 nebo vyšší.

Popis

Metoda; zavádí spojení se specifikovaným Internet hostitelem pomocí použití specifikovaného TCP portu (musí být 1024 nebo vyšší) a udává **`true`** nebo **`false`** v závislosti na úspěšném zavedení spojení. Jestliže neznáte číslo portu vašeho Internet hostitelského stroje, kontaktujte vašeho síťového administrátora. Jestliže je používán Flash Netscape plug-in nebo ActiveX control, hostitel specifikovaný argumentu musí mít stejnou subdoménu jako hostitel, ze kterého byla stažena animace.

Pokud pro argument **host** specifikujete **`null`**, kontaktovaný hostitel bude ten hostitel, kde sídlí animace volající XML**`Socket.connect`**. Například, jestliže animace byla stažena z http://www.yoursite.com, specifikující **`null`** pro argument hostitel je stejné jako vložení IP adresy pro xxx.yoursite.com.

Jestliže XML**`Socket.connect`** udává hodnotu **`true`**, počáteční fáze procesu spojení je úspěšná; později je vyvolána metoda XML**`Socket.onLoad`** pro určení, zda je konečné spojení úspěšné nebo neúspěšné. Jestliže XML**`Socket.connect`** udává **`false`**, spojení nemohlo být zavedeno.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad používá XML**`Socket.connect`** pro spojení s hostitelem, kde sídlí animace a používá **`trace`** pro zobrazení udané hodnoty indikující úspěch nebo neúspěch spojení:

```
function myOnConnect(success){
       if (success){
              trace ("Connection succeeded!")
       }else {
              trace ("Connection failed!")
       }
}
socket = new XMLSocket()
socket.onConnect =myOnConnect
if (!socket.connect(null,2000)){
       trace ("Connection failed!")
}
```

Viz také

`function`
`XMLSocket.onLoad`

XMLSocket.onClose

Syntaxe

`myXMLSocket.onClose();`

Argumenty

Žádné.

Popis

Metoda; funkce zpětného volání, která je vyvolána, když je serverem zavřeno otevřené spojení. Daná implementace této metody neprovádí žádné akce. Pro přepsání dané implementace musíte připojit funkci obsahující vaše vlastní akce.

Přehrávač

Flash 5 nebo novější.

Viz také

`function`
`XMLSocket.onLoad`

XMLSocket.onConnect

Syntaxe

myXMLSocket.onConnect(success);

Argumenty

success Booleánská hodnota indikující, zda bylo zásuvkové spojení úspěšně zavedeno (**true** nebo **false**).

Popis

Metoda; funkce zpětného volání vyvolána Flash Přehrávačem, když požadavek na spojení iniciovaný pomocí metody XML**Socket.connect** byl úspěšný nebo neúspěšný. Jestliže bylo spojení úspěšně, argument **success** je **true**; jinak je argument **success false**.

Daná implementace této metody neprovádí žádné akce. Pro přepsání dané implementace musíte určit funkci obsahující vaše vlastní akce.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad ilustruje proces specifikace nahrazení funkce pro metodu **onConnect** v jednoduché chat aplikaci.

Funkce kontroluje, ke které obrazovce se uživatelé dostanou v závislosti na tom, zda je úspěšně zavedeno spojení. Jestliže je spojení úspěšně zavedeno, uživatelé se dostanou k hlavní chat obrazovce na snímku popsaném **startChat**. Jestliže spojení není úspěšné, uživatelé jdou na obrazovku, kde je informace na popisku **connectionFailed**.

```
function myOnConnect(success){
       if (success){
              gotoAndPlay("startChat")
       }else {
              gotoAndStop("connectionFailed")
       }
}
```

Po vytvoření XML**Socket** objektu pomocí použití konstrukční metody, scénář instaluje metodu **onConnect** použitím operátoru určení (assignment):

```
socket = new XMLSocket()
socket.onConnect = mujOnConnect
```


Nakonec je inicializováno spojení. Jesltiže **connect** udává **false**, animace je poslána přímo do snímku popsaného **connectionFailed** a **onConnect** už není více vyvolán. Jestliže connect udává **true**, animace skočí na snímek popsaný **waitForConnect**, což je obrazovka „Prosím čekejte“. Animace zůstává na snímku **waitForConnect**, dokud není vyvolán ovladač **onConnect**, což se stane na stejném bodě v budoucnosti v závislosti na síťové latentnosti.

```
if (!socket.connect(null,2000)){
       gotoAndStop("connectionFailed")
}else {
       gotoAndStop("waitForConnection")
}
```

Viz také

XMLSocket.connect
function

XMLSocket.onXML

Syntaxe

myXMLSocket.**onXML(objekt)**;

Argumenty

objekt Instance XML objektu obsahující analyzovaný XML dokument obdržený ze serveru.

Popis

Metoda; funkce zpětného volání vyvolaná Flash Přehrávačem, když specifikovaný XML objekt obsahující XML dokument přejde přes otevřené XMLSocket spojení. XMLSocket spojení může být použito pro transfer neomezeného počtu XML dokumentů mezi klientem a serverem. Každý dokument je ukončen nulovým bytem. Když Flash Přehrávač obdrží nulový byte, analyzuje všechny XML obdržené od předchozího nulového bytu nebo od zavedení spojení, pokud je toto první obdržená zpráva. S každou sérií analyzovaných XML se zachází jako s jedním XML dokumentem a je propuštěna do metody **onXML**.

Daná implementace této metody neprovádí žádné akce. Pro přepsání dané implementace musíte určit funkci obsahující akce, které definujete.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující funkce přepisuje danou implementaci metody onXML v jednoduché chat aplikaci. Funkce **myOnXML** instruuje chat aplikaci, aby rozpoznala jednotlivý XML prvek, MESSAGE, v následujícím formátu:

```
<MESSAGE USER = "John" TEXT = "AHOJ, jmenuji se John!"/>.
```

Ovladač naXML musí být nejprve instalován v XMLSocket objektu následovně:

```
socket.onXML = myOnXML;
```

Předpokládá se, že funkce **displayMessage** je funkce definovaná uživatelem, která uživateli zobrazí obdrženou zprávu.

```
function myOnXML(doc){
   var e = doc.firstChild;
   if (e != null && e.nodeName == "MESSAGE"){
       displayMessage(e.attributes.user, e.attributes.text);
   }
}
```

Viz také

`function`

XMLSocket.send

Syntaxe

myXMLSocket`.send(objekt);`

Argumenty

objekt XML objekt nebo jiná data, která se mají přenést na server.

Popis

Metoda; konvertuje XML objekt nebo data specifikovaná v argumentu objekt na řetězec a přenáší je na server, následované nulovým bytem. Jestliže je objekt XML objekt, řetězec je XML textová reprezentace XML objektu. Operace posílání je asynchronní; vrací se okamžitě, ale data mohou být přenesena později. Metoda XML**Socket.send** neudává hodnotu indikující, zda byla data úspěšně přenesena.

Jestliže **myXMLSocket** objekt není spojen se serverem (použitím XML**Socket.connect**), operace X**MLSocket.send** bude neúspěšná.

Přehrávač

Flash 5 nebo novější.

Příklad

Následující příklad ilustruje, jak byste mohli specifikovat jméno uživatele a heslo pro poslání XML objektu **myXML** do serveru:

```
var myXML = new XML();
var myLogin = myXML.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
myXML.appendChild(myLogin);
myXMLSocket.send(myXML);
```

Viz také

`XMLSocket.connect`

_xmouse

Syntaxe

instancejméno`._xmouse`

Argumenty

instancejméno Jméno movie klip instance.

Popis

Vlastnost (pouze pro čtení); udává souřadnici **x** pozice myši.

Přehrávač

Flash 5 nebo novější.

Viz také

`Mouse (object)`
`_ymouse`

_xscale

Syntaxe

```
instancejméno._xscale
instancejméno._xscale = procento;
```

Argumenty

procento Procentní hodnota specifikující procento pro horizontální měřítko animace. Daná hodnota je 100.

instancejméno Jméno movie klip instance.

Popis

Vlastnost; udává horizontální měřítko (procento) movie klipu jako aplikované z registračního bodu movie klipu. Daný registrační bod je (0,0).

Měřítko lokálního systému souřadnic ovlivňuje nastavení vlastností **_x** a **_y**, které jsou definovány v celých pixelech. Například, jestliže je movie klip zmenšen na 50%, nastavení vlastnosti **_x** posune objekt v movie klipu o polovinu počtu pixelů, na rozdíl od toho, kdyby bylo movie na 100%.

Přehrávač

Flash 4 nebo novější.

Viz také

_yscale

_y

Syntaxe

```
instancejméno._y
instancejméno._y = celé číslo;
```

Argumenty

celé číslo Lokální souřadnice y movie klipu.

instancejméno Jméno movie klip instance.

Popis

Vlastnost; nastavuje souřadnici **y** movie relativně k lokálním souřadnicím rodičovského movie klipu. Jestliže je movie klip na hlavní Časové ose, potom jeho systém souřadnic odpovídá levému hornímu rohu Scény jako (0,0). Jestliže je movie klip uvnitř jiného movie klipu, který je transformován, movie klip je v lokálním systému souřadnic vloženého movie klipu. Tedy, pro movie klip otočený o 90° proti směru hodinových ručiček, zdědí děti movie klipu systém souřadnic, který je otočen o 90° proti směru hodinových ručiček. Souřadnice movie klipu odpovídají pozici registračního bodu.

Přehrávač

Flash 3 nebo novější.

Viz také

_yscale

_ymouse

Syntaxe

```
instancejméno._ymouse
```

Argumenty

instancejméno Jméno movie klip instance.

Popis

Vlastnost (pouze pro čtení); indikuje souřadnici **y** pozice myši.

Přehrávač

Flash 5 nebo novější.

Viz také

Mouse (object)
_xmouse

_yscale

Syntaxe

instancejméno`._yscale`
instancejméno`._yscale = procento;`

Argumenty

procento Procentní hodnota specifikující procento pro vertikální měřítko animace. Daná hodnota je 100.

instancejméno Jméno movie klip instance.

Popis

Vlastnost; nastavuje vertikální měřítko (procento) movie klipu, jako aplikované z registračního bodu movie klipu. Daný registrační bod je (0,0).

Měřítko lokálního systému souřadnic ovlivňuje nastavení vlastností **_x** a **_y**, které jsou definovány v celých pixelech. Například, jestliže je movie klip zmenšen na 50%, nastavení vlastnosti **_x** posune objekt v movie klipu o polovinu počtu pixelů, na rozdíl od toho, kdyby byla animace na 100%.

Přehrávač

Flash 4 nebo novější.

Viz také

_x
_y

DODATEK A
Nadřazenost a asociativita operátorů

Seznam Operátorů

Toto je seznam všech ActionScript operátorů a jejich asociativity, podle nadřazenosti od nejvyšší po nejnižší.

Operátor	Popis	Asociativita
+	Unary plus	Right to left
-	Unary minus	Right to left
~	Bitwise one s complement	Right to left
!	Logical NOT	Right to left
not	Logical NOT (Flash 4 style)	Right to left
++	Post-increment	Left to right
--	Post-decrement	Left to right
()	Function call	Left to right
[]	Array element	Left to right
.	Structure member	Right to left
++	Pre-increment	Right to left
--	Pre-decrement	Right to left
new	Allocate object	Right to left
delete	Deallocate object	Right to left
typeof	Type of object	Right to left
void	Returns undefined value	Left to right
*	Multiply	Left to right
/	Divide	Left to right
%	Modulo	Left to right
+	Add	Left to right
add	String concatenation (formely &)	Left to right
-	Subtract	Left to right
<<	Bitwise Left Shift	Left to right
>>	Bitwise Right Shift	Left to right
>>>	Bitwise Right Shift (Unsigned)	Left to right
<	Less than	Left to right
<=	Less than or equal to	Left to right
>	Greater than	Left to right
>=	Greater than or equal to	Left to right

Operátor	Popis	Asociativita
>=	Greater than or equal to	Left to right
lt	Less than (string version)	Left to right
le	Less than or equal to (string version)	Left to right
gt	Greater than (string version)	Left to right
ge	Greater than (string version)	Left to right
==	Equal	Left to right
!=	Not equal	Left to right
eq	Equal (string version)	Left to right
ne	Not equal (string version)	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
and	Logical AND (Flash 4)	Left to right
||	Logical OR	Left to right
or	Logical OR (Flash 4)	Left to right
?:	Conditional	Right to left
=	Assignment	Right to left
"*=, /=, %=, += -=, &=, |=, ^=, <<=, >>=, >>>="	Compound assignment	Right to left
,	Multiple evaluation	Left to right

DODATEK B

Klávesy Klávesnice a Hodnoty Kódů Kláves

Následující tabulky ukazují seznam všech standartních kláves klávesnice a jim odpovídající hodnoty kódů, které jsou používány v ActionScriptu. Pro více informací se podívejte na objekt Key v kapitole 7, "ActionScript Slovníku".

Písmena od A do Z a standartní číslice 0 až 9

Klávesa písmeno, nebo číslice	Kód klávesy
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Klávesy na numerické klávesnici

Numerická klávesa	Kód klávesy
Numklávesa 0	96
Numklávesa 1	97
Numklávesa 2	98
Numklávesa 3	99
Numklávesa 4	100
Numklávesa 5	101
Numklávesa 6	102
Numklávesa 7	103
Numklávesa 8	104
Numklávesa 9	105
Násobení	106
Plus	107
Enter	108
Odečítání	109
Desetina	110
Dělení	111

Klávesy funkcí

Numerická klávesa	Kód klávesy
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123

Ostatní klávesy

Klávesa	Kód klávesy
Backspace	8
Tab	9
Clear	12
Enter	13
Shift	16
Control	17
Alt	18
Caps Lock	20
Esc	27
Spacebar	32
Page Up	33
Page Down	34
End	35
Home	36

Klávesa	Kód klávesy
Left Arrow	37
Up Arrow	38
Right Arrow	39
Down Arrow	40
Insert	45
Delete	46
Help	47
Num Lock	144
;:	186
= +	187
-_	189
/?	191
~	192
[{	219
|	220
{[	221
' "	222

DODATEK C
Chybová hlášení

Následující tabulka obsahuje seznam hlášení o chybách udaných Flash kompilátorem. U každé zprávy je poskytnuto vysvětlení, které vám má pomoci při odstraňování problémů ve vašich movie souborech.

Chybové hlášení	Kód klávesy
`Property <property>does not exist`	Došlo k setkání s vlastností, která neexistuje. Například `x = zelená` je neplatná, protože `_zelená` není žádná vlastnost.
`Operator <operator>must be followed by an operand`	Došlo k setkání s operátorem bez operandu. Například `x = 1 +` vyžaduje operand za operátorem `+`. Operátor je následován neplatným operandem. Například `trace(1+)` je syntakticky nesprávně.
`Syntax error`	Tato zpráva je vydána, kdykoliv dojde k setkání s nespecifikovanou chybou syntaxe.
`Expected a field name after '.' operator`	Když používáte syntaxi **objekt.pole**, Musíte specifikovat platné jméno pole.

`Expected <token>`	Došlo na neplatný nebo neočekávaný symbol. Například v syntaxi dole, symbol `foo` není platný. Očekávaný symbol je `while`.
	`do {`
	`trace (i)`
	`} foo (i < 100)`

`Initializer list must be terminated by <terminator>`	V inicializačním seznamu objektu nebo pole chybí ukončovací závorka `]` nebo `}`.
`Identifier expected`	Došlo k setkání s neočekávaným symbolem na místě identifikátoru. V příkladu dole není `3` platný identifikátor. `var 3 = 4;`
`The JavaScript '<construct> 'construct is not supported`	Došlo k setkání s JavaScript konstrukcí, která není podporováno ActionScript. Tato zpráva se objeví, jestliže jsou použity jakékoliv z následujících JavaScript konstrukcí: `void, switch, try, catch` nebo `throw`.
`Left side of assignment operator must be variable or property`	Byl použit operátor určení, ale na levé straně určení nebyla legální proměnná nebo vlastnost.
`Statement block must be terminated by '}'`	Skupina výrazů byla deklarována ve složených závorkách, ale chybí ukončovací závorka.
`Event expected`	Byl deklarován ovladač `On(MouseEvent)` nebo `onClipEvent`, ale nebyla specifikována událost nebo došlo k setkání s neočekávaným symbolem tam, kde se měla objevit událost.
`Invalid event`	Scénář obsahuje neplatnou událost myši nebo klipu. Seznam platných událostí myši a klipu viz položky On(MouseEvent) a onClipEvent v kapitole slovníku ActionScript.
`Key code expected`	Musíte specifikovat kód klávesy. Seznam kódů kláves viz Dodatek B.
`Invalid key code`	Specifikovaný kód klávesy neexistuje.
`Trailing garbage found`	Scénář nebo výraz byl analyzován správně, ale obsahoval dodatečné stopovací znaky, které nemohly být analyzovány.

Chybové hlášení	Kód klávesy

`Illegal function`	Byla použita deklarace jmenované funkce jako výraz. Deklarace jmenované funkce musí být příkaz.Platné:
	`function sqr (x){`
	`return x * x;`
	`}`
	`Neplatné: var v = function sqr (x){`
	`return x * x;`
	`}`

`Function name expected`	Jméno specifikované pro tuto funkci není platné jméno funkce.
`Parameter name expected`	V deklaraci funkce bylo očekáváno jméno parametru (argumentu), ale došlo k setkání s neočekávaným symbolem.
`else'encountered without matching'if'`	Došlo k setkání s příkazem `else`, ale před ní se neobjevilo žádné `if`. Můžete použít `else` pouze ve spojení s příkazem `if`.
`Scene type error`	Argument scény akce `gotoAndPlay`, `gotoAndStop` nebo `ifFrameLoaded` byl špatného typu. Argument scény musí být řetězcová konstanta.
`Internal error`	V ActionScript kompilátoru se objevila vnitřní chyba. Prosím pošlete FLA soubor, který generoval tuto chybu do Macromedia s detailními instrukcemi, jak reprodukovat zprávu.
`Hexadecimal digits expected after 0x`	Došlo k setkání se sekvencí 0x, ale sekvence nebyla následována platnými hexadecimálními číslicemi.
`Error opening #include file`	Byla zde chyba otevírající zahrnutý soubor s příkazem `include`. Chyba se mohla objevit proto, že soubor nebyl přítomný nebo v důsledku chyby disku.
`Malformed #include directive`	Příkaz `include` nebyl napsán správně. Příkaz `include` musí používat následující syntaxi: `#include „nějakýsoubor.as“`
`Multi-line comment was not terminated`	Několika řádkový komentář začal `s /*` a chybí zavírací `*/` tag.
`String literal was not properly terminated`	U písmenného řetězce začínajícího otevírajícím citačním znaménkem (jednoduchým nebo dvojitým) chybí zavírající citační znaménko.
`Function <function>takes <count> parameters`	Byla volána funkce, ale došlo k setkání s neočekávaným počtem parametrů.
`Property name expected in GetProperty`	Byla volána funkce `getProperty`, ale druhý argument nebylo jméno movie klip vlastnosti.
`Parameter <parameter>cannot be declared multiple times`	Jméno parametru se objevilo několikrát v parametru deklarace funkce. Všechna jména parametrů musí být jedinečná.
`Variable <variable>cannot be declared multiple times`	Jméno proměnné se objevilo několikrát v příkazu `var`. Všechna jména proměnných v jednom příkazu `var` musí být jedinečná.
`'on' handlers may not be nested within other 'on' handlers`	Ovladač `on` byl deklarován uvnitř jiného ovladače `on`. Všechny ovladače se musí objevovat v nejvyšší úrovni seznamu akce.
`Statement must appear within on handler`	V akcích pro instanci tlačítka byl deklarován příkaz bez obklopujícího bloku on. Všechny akce pro instanci tlačítka se musí objevit uvnitř bloku on.
`Statement must appear within onClipEvent handler`	V akcích pro movie klip instanci byl deklarován příkaz bez obklopujícího bloku `onClipEvent`. Všechny akce pro movie klip instanci se musí objevit uvnitř bloku `onClipEvent`.

Chybové hlášení	Kód klávesy
`Mouse events are permitted only for button instances`	Ovladač události tlačítka byl deklarován v seznamu akcí snímku nebo v seznamu akcí movie klip instance. Události tlačítek jsou povoleny pouze v seznamech akcí instancí tlačítek.
`Clip events are permitted only for movie clip instances`	Ovladač události klipu byl deklarován v seznamu akcí snímku nebo v seznamu akcí instance tlačítka. Události klipu jsou povoleny pouze v seznamech akcí movie klip instancí.