

3부

레퍼런스

3부에서는 액션스크립트에서 지원되는 내장 클래스, 객체, 함수, 속성, 그리고 이벤트 핸들러의 사용법을 설명한다. 여러분이 3부의 내용을 규칙적으로 참조하다 보면 일상적인 프로그래밍 업무에서 특별한 작업을 이루어낼 수 있을 것이다.

- 액션스크립트 레퍼런스

액션스크립트 레퍼런스

이제 액션스크립트에서 제공되는 모든 클래스와 객체에 대해 알아보자. '3부. 레퍼런스'에서는 각 클래스와 객체의 일반적인 용도, 사용법, 속성, 메소드, 이벤트 핸들러에 대해 다룬다. 또한 (어떤 클래스나 객체에도 포함되지 않고 무비의 어느 곳에서나 독립적으로 사용할 수 있는) 전역 함수와 전역 속성에 대해서도 다룰 것이다.

레퍼런스에 수록된 내용은 모두 알파벳 순서대로 정리하였다. 예를 들어 전역 함수인 `duplicateMovieClip()`은 `Date` 클래스보다 뒤에 나온다(전역 함수만을 따로 모아서 수록하지 않았다). 하지만 각 아이템별로 클래스, 객체, 전역 함수 등을 확실히 구분할 수 있도록 만들었다. `duplicateMovieClip()` 같은 경우는 전역 함수와 무비 클립 메소드 두 가지로 쓰이므로, 'duplicateMovieClip() 전역 함수'와 'MovieClip.duplicateMovieClip() 메소드'와 같이 표기하여 전역 함수인지 메소드인지 확실히 구분할 수 있도록 하였다. 필요한 내용을 어디에서 찾아야 할지 잘 모르겠다면 이 책 맨 뒤에 있는 '찾아보기'를 참조하기 바란다.

전역 함수

전역 함수는 무비 전반에 걸쳐 사용할 수 있는 내장 함수이다. 전역 함수는 무비에 있는 모든 프레임이나 버튼, 이벤트 핸들러에서 호출할 수 있다(메소드는 특정 객체에 대한 레퍼런스를 통해 호출해야 한다).

[표 R-1]은 플래시 5 액션스크립트에서 사용할 수 있는 전역 함수 목록이다.

[표 R-1] 액션스크립트 전역 함수

Boolean()	call()	Date()
duplicateMovieClip()	escape()	eval()
fscommand()	getProperty()	getTimer()
getURL()	getVersion()	gotoAndPlay()
gotoAndStop()	#include	int()*
isFinite()	isNaN()	loadMovie()
loadMovieNum()	loadVariables()	loadVariablesNum()
maxscroll	newline	nextFrame()
nextScene()	Number()	parseFloat()
parseInt()	play()	prevFrame()
prevScene()	print()	printAsBitmap()
printAsBitmapNum()	printNum()	random()*
removeMovieClip()	scroll	setProperty()
startDrag()	stop()	stopAllSounds()
stopDrag()	String()	targetPath()
tellTarget()*	toggleHighQuality()*	trace()
unescape()	unloadMovie()	unloadMovieNum()
updateAfterEvent()		

* 플래시 5에서는 더 이상 쓰이지 않음

전역 속성

전역 속성은 전역 함수와 마찬가지로 무비에 있는 모든 스크립트에서 사용할 수 있는 속성이다. 전역 속성은 어떤 코드에서 필요한 정보라도 포함하며, 특정 무비 클립이나 무비뿐만 아니라 플래시 플레이어 전체에도 영향을 미친다.

[표 R-2]는 플래시 5에서 사용할 수 있는 전역 함수의 목록이다.

[표 R-2] 액션스크립트 전역 속성

속성 이름	설명
_focusrect	키보드를 통해 활성화되는 버튼의 하이라이트 상태
_highquality	플레이어의 렌더링 화질*
Infinity	숫자형에서 양의 무한대를 나타내는 상수
-Infinity	숫자형에서 음의 무한대를 나타내는 상수
_leveln	플레이어의 문서 레벨
NaN	잘못된 숫자 데이터를 나타내는 숫자형 값(Not-A-Number)
_quality	플레이어의 렌더링 화질
_root	현재 레벨의 메인 무비 타임라인에 대한 레퍼런스
_soundbuftime	미리 가져올 스트립 사운드 분량(초 단위)
\$version*	플래시 플레이어 버전

* 플래시 5에서는 더 이상 쓰이지 않음

내장 클래스와 객체

여기서는 클래스, 객체, 인스턴스라는 용어를 '12장. 객체와 클래스'에서 정의한 것과 같은 의미로 구분하여 사용한다. 액션스크립트의 내장 클래스는 무비를 제어하고 데이터를 조작하기 위한 객체를 만드는 데 쓰인다. 액션스크립트의 내장 클래스에는 Array, Boolean, Color, Date, MovieClip, Number, Object, Sound, String, XML, XMLNode, XMLSocket이 있다. 어떤 클래스의 인스턴스를 만들려면 new 연산자와 클래스의 생성자 함수를 함께 사용하면 된다. 예를 들어 Color 클래스의 인스턴스를 만들 때는 다음과 같은 식으로 Color 생성자를 이용한다.

```
myColor = new Color(_root);
```

클래스를 설명할 때는 '생성자' 부분에서 그 클래스에 속하는 객체를 만드는 방법(각 클래스의 생성자 함수를 호출하기 위한 문법)을 설명한다. 각 클래스의 객체에서 사용할 수 있는 속성, 메소드, 이벤트 핸들러와 클래스의 용도, 일반적인 사용법도 모두 수록하였다. 클래스에 따라 클래스 인스턴스가 아닌 클래스 생성자 자체를

통해서 액세스할 수 있는 속성이나 메소드가 들어있는 것도 있다. 이러한 메소드와 속성은 ‘클래스 메소드’와 ‘클래스 속성’으로 표기하였다. 클래스의 속성, 메소드, 이벤트 핸들러에 대한 자세한 설명은 각 클래스에 대한 일반적인 설명 뒤에 알파벳순으로 수록하였다.

액션스크립트의 특수 내장 객체(Arguments, Key, Math, Mouse, Selection)도 알파벳순으로 수록하였지만, ‘객체’라는 단어가 붙어 있으면 내장 객체라고 보면 된다(예: Math 객체). 생성자 함수를 이용하여 서로 다른 객체를 여러 개 만들 수 있는 일반적인 클래스와는 달리 이 특수 내장 객체에서는 인스턴스를 만들 수 없다(즉 new 연산자를 이용하여 객체를 만들 수 없다). 이 객체들은 몇 가지 연관된 기능을 하나로 모아두기 위한 용도로 만든 객체이다. 대표적인 예로 Math 객체를 이용하여 자주 쓰이는 수학 함수나 상수를 간편하게 사용할 수 있다는 점을 들 수 있다.

제목 설명

레퍼런스에 수록된 각 아이템을 설명하기 위해 [표 R-3]에 나온 것과 같은 제목을 사용한다.

[표 R-3] 레퍼런스에서 사용하는 제목

제목	설명
버전	해당 아이템이 언제부터 액션스크립트에 추가되었는지 설명하는 부분. 하위 호환성 및 폐기 여부도 설명한다.
문법	아이템을 사용할 때 적용되는 일반적인 문법. 프로그래머가 별도로 입력해야 하는 부분은 이탤릭으로 표기하였다.
인자	메소드와 함수에만 적용된다. 함수나 메소드를 호출하는 데 필요한 매개변수를 설명하는 부분
리턴 값	메소드와 함수에만 적용된다. 메소드나 함수에 리턴 값이 있는 경우에 그 리턴 값에 대한 설명을 수록하였다. 리턴 값이 없으면 이 부분은 생략된다.
액세스	속성에만 적용된다. 속성 값이 읽기 전용인지 아니면 읽기 쓰기가 모두 가능한지 알려주는 부분
설명	그 아이템이 어떤 식으로 동작하는지, 그리고 실전에서 어떤 식으로 사용할 수 있는지 설명하는 부분

제목	설명
주의 사항	아이템과 관련해서 알아둬야 할 내용을 설명하는 부분
버그	해당 아이템과 관련하여 현재까지 알려진 내용을 설명하는 부분
예제	아이템을 사용하는 샘플 코드
참조	해당 아이템과 관련된 내용

알파벳순 레퍼런스

이제부터 액션스크립트의 객체와 클래스에 대해 하나씩 알아보자. 연산자나 선언문과 같이 '3부. 레퍼런스'에서 따로 설명하지 않는 부분은 뒤에 있는 '찾아보기'에서 찾아보기 바란다.

arguments 객체

현재 함수 및 매개변수 액세스

버전	플래시 5
문법	<code>arguments[elem]</code> <code>arguments.propertyName</code>
속성	
<i>callee</i>	실행 중인 함수에 대한 레퍼런스
<i>length</i>	함수에 전달된 매개변수의 개수

설명

arguments 객체는 각 함수에 arguments라는 지역 변수 형태로 저장되며 함수 실행 중에만 사용할 수 있다. arguments는 배열인 동시에 객체이다. 우선 현재 실행 중인 함수에 전달된 매개변수는 arguments에 인덱스가 있는 배열 형태로 저장된다. 예를 들면 arguments[0]은 첫 번째 매개변수, arguments[1]은 두 번째 매개변수와 같은 식으로 저장된다. 또한 현재 함수를 확인하거나 다시 호출하는 데 쓰이는 callee 속성도 arguments 객체에 저장된다.

참조

‘9장. 함수’의 ‘arguments 객체’

arguments.callee 속성

실행 중인 함수를 가리키는 레퍼런스

버전 플래시 5

문법 `arguments.callee`

액세스 읽기/쓰기

설명

callee 속성에는 현재 실행 중인 함수에 대한 레퍼런스가 저장된다. 이 레퍼런스를 이용하면 실행 중인 함수를 다시 호출하거나 비교 연산자를 이용하여 현재 함수를 확인할 수 있다.

예제

```
function someFunction () {
    trace(arguments.callee == someFunction);    // true가 출력된다.
}

// 익명 재귀함수
countToTen = function () {
    i++;
    trace(i);
    if (i < 10) {
        arguments.callee();
    }
};
```

arguments.length 속성

인자로 전달된 매개변수의 개수

버전 플래시 5

문법 `arguments.length`

액세스 읽기/쓰기

설명

`length` 속성에는 실행 중인 `arguments` 배열의 원소 개수가 정수 형태로 저장된다. 이 값은 함수에 전달된 매개변수의 개수와 같다.

예제

`arguments` 배열의 `length` 속성을 이용하여 함수에 전달된 매개변수의 개수를 확인할 수 있다. 아래 코드에서는 `someFunction()`이라는 함수에 두 개의 인자가 전달되었는지 확인한다. 이 예제를 조금 고쳐서 매개변수의 데이터형이 정확한지 확인하는 코드도 직접 만들어보자(힌트: `typeof` 연산자를 이용하면 된다).

```
function someFunction (y, z) {
    if (arguments.length != 2) {
        trace("Function invoked with wrong number of parameters");
        return;
    }
    // 함수에서 실제 작업을 처리하는 부분
}
```

Array 클래스

순서가 정해진 데이터 리스트

버전 플래시 5

생성자 `new Array()`
`new Array(len)`
`new Array(element0, element1, element2, ...elementn)`

인자

len 새로운 배열의 길이를 지정하기 위한 0 이상의 정수

element0,...,elementn

배열의 원소로 대입할 한 개 이상의 초기 값 리스트

속성

length 배열에 들어갈 원소의 개수(비어있는 원소 포함)

메소드

<i>concat()</i>	원래 있는 배열에 원소를 추가하여 새로운 배열을 만드는 메소드
<i>join()</i>	배열을 문자열로 변환하는 메소드
<i>pop()</i>	배열의 마지막 원소를 제거하고 그 값을 리턴하는 메소드
<i>push()</i>	배열의 끝에 한 개 이상의 새로운 원소를 추가하는 메소드
<i>reverse()</i>	배열에 있는 원소의 순서를 거꾸로 뒤집는 메소드
<i>shift()</i>	배열의 첫 번째 원소를 삭제하고 그 값을 리턴하는 메소드
<i>slice()</i>	원래 있던 배열의 일부를 이용하여 새로운 배열을 만드는 메소드
<i>sort()</i>	배열의 원소를 주어진 규칙에 따라 정렬하는 메소드
<i>splice()</i>	배열에 원소를 추가하거나 배열에서 원소를 제거하는 메소드
<i>toString()</i>	배열의 각 원소를 쉼표 구분자를 사용하는 문자열로 변환하는 메소드
<i>unshift()</i>	한 개 이상의 원소를 배열의 맨 앞에 추가하는 메소드

설명

Array 클래스의 속성과 메소드를 이용하면 배열 객체의 원소로 저장된 데이터를 조작할 수 있다. 배열의 정의와 사용법 및 배열에서 쓰이는 여러 용어의 정확한 의미에 대해 자세히 알고 싶다면 '11장. 배열'을 참조하기 바란다. 배열의 원소를 액세스하는 데 쓰이는 연산자인 []와 . 연산자에 대한 내용은 '5장. 연산자' 부분에 나와 있다.

주의 사항

Array 생성자를 호출할 때 하나의 정수 인자만 사용하면 그 인자는 첫 번째 원소의 값으로 저장되는 것이 아니라 새로 만들 배열의 길이로 인식된다. 생성자에 두 개 이상의 인자를 전달하거나 숫자가 아닌 인자를 전달하면 그 인자가 배열 원소의 초기 값으로 사용되며 배열의 길이는 주어진 인자의 개수에 의해 결정된다.

Array.concat() 메소드

기존 배열을 확장하여 새로운 배열을 만든다.

버전 플래시 5

문법 array.concat(value1, value2, value3,...valuen)

인자*value1,...valuen*

array의 맨 뒤에 새로운 원소로 추가할 표현식의 리스트

리턴 값

array의 모든 원소 뒤에 value1,...valuen의 모든 원소를 추가한 새로운 배열

설명

concat() 메소드는 어떤 배열의 맨 뒤에 새로운 원소를 추가한 새로운 배열을 리턴한다. 이 때 원본 배열은 바뀌지 않는다. 원본 배열을 바로 고칠 때는 push(), splice() 또는 shift() 메소드를 사용한다.

concat() 메소드의 인자로 배열을 사용하면 그 배열의 각 원소가 원래 배열에 추가된다. 즉 arrayX.concat(arrayY)라고 하면 arrayY의 각 원소를 arrayX의 뒷부분에 추가한 새로운 배열이 만들어진다. 이렇게 만들어진 배열의 length 속성은 arrayY.length + arrayX.length가 된다. 하지만 중첩된 배열의 원소까지 모두 하나씩 배열에 추가되지는 않는다(중첩된 배열은 배열 형태로 추가된다).

예제

```
// 새로운 배열을 만든다.
myListA = new Array("apples", "oranges");

// myListB를 ["apples", "oranges", "bananas"]로 설정한다.
myListB = myListA.concat("bananas");

// 또 다른 배열을 만든다.
myListC = new Array("grapes", "plums");

// myListD를 ["apples", "oranges", "bananas", "grapes", "plums"]로
// 설정한다.
myListD = myListB.concat(myListC);

// myListA를 ["apples", "oranges", "bananas"]로 설정한다.
myListA = myListA.concat("bananas");

// 새로운 배열을 만든다.
settings = ["on", "off"];
```

```
// 중첩된 배열이 포함된 배열을 덧붙인다.
options = settings.concat(["brightness", ["high", "medium", "low"]]);
// options는 ["on", "off", "brightness", "high", "medium", "low"]가
// 아니라
// ["on", "off", "brightness", ["high", "medium", "low"]]가 된다.
```

참조

Array.push(), Array.shift(), Array.splice(), 11장의 ‘concat() 메소드’

Array.join() 메소드

배열을 문자열로 변환한다.

버전 플래시 5

문법 `array.join()`
`array.join(delimiter)`

인자

delimiter 새로 만드는 문자열에서 각 원소 사이에 집어넣을 문자열. 이 인자를 전달하지 않으면 쉼표를 기본값으로 사용한다.

리턴 값

배열의 모든 원소를 문자열로 변환하고 각 원소 사이에 delimiter를 집어넣은 문자열

설명

join() 메소드는 다음과 같은 방법으로 배열의 모든 원소를 합쳐서 만든 문자열을 리턴한다.

1. 배열의 각 원소를 문자열로 변환한다(비어있는 원소는 비어있는 문자열로 변환된다).
2. 1단계에서 문자열로 변환한 각 원소 뒤에 delimiter를 추가한다(마지막 원소는 제외).
3. 2단계를 거친 모든 원소(문자열)를 하나의 긴 문자열로 합친다.

원소 중에 배열이 있으면 그 배열은 toString() 메소드를 통해 문자열로 변환되므로 중첩된 배열은 join()을 호출할 때 인자로 전달된 구분자 대신 쉼표로 구분된다.

예제

```
fruit = new Array("apples", "oranges", "bananas", "grapes", "plums");
// fruitString을 "apples,oranges,bananas,grapes,plums"로 설정한다.
fruitString = fruit.join();
// fruitString을 "apples-oranges-bananas-grapes-plums"로 설정한다.
fruitString = fruit.join("-");
```

참조

Array.toString(), String.split(), 11장의 'join() 메소드'

Array.length 속성

배열에 있는 원소의 개수

버전	플래시 5
문법	array.length
액세스	읽기/쓰기

설명

length 속성은 배열에 있는 원소의 개수를 나타내는 0 이상의 정수이다. 원소가 하나도 없는 배열의 길이는 0이고 원소가 두 개인 배열의 길이는 2이다. 첫 번째 원소의 인덱스 번호는 0이기 때문에 length 값은 배열의 마지막 원소의 인덱스 번호에 1을 더한 값이 된다.

어떤 배열의 length 속성은 비어있는 원소(null 또는 undefined 값이 저장되어 있는 원소)를 포함하여 배열에 번호 인덱스를 사용하는 원소가 몇 개 있는지를 나타낸다. 예를 들어 0, 1, 2, 9번 원소에는 값이 들어있지만 3번부터 8번 원소까지는 비어있는 배열이 있을 수도 있다. 이 배열에서 어떤 값이 저장되어 있는 원소는 네 개뿐이지만, 10개의 원소(0번부터 9번)가 들어갈 자리를 차지하고 있기 때문에 배열의 길이는 10이다.

배열의 length 값을 바꾸면 배열 원소의 개수도 바뀐다. length 값을 증가시키면 배열의 뒤에 비어있는 원소가 추가되고 length 값을 줄이면 배열의 뒤에서 그만큼 원소가 삭제된다. 또한 Array 클래스 메소드를 이용하여 배열에 원소를 추가하거나 배열에서 원소를 제거하면 length 속성 값도 자동으로 바뀐다. length 속성으로는 숫자 인덱스의 개수만을 알 수 있다. 배열 속성과 같이 이름이 있는 원소의 개수는 length 속성에 포함되지 않는다.

예제

```
myList = new Array("one", "two", "three");
trace(myList.length); // 3이 출력된다.

// 배열의 원소별로 루프를 돌린다.
for (var i = 0; i < myList.length; i++) {
    trace(myList[i]);
}
```

참조

11장의 '배열의 크기 알아내기'

Array.pop() 메소드

배열의 마지막 원소를 제거한다.

버전 플래시 5

문법 `array.pop()`

리턴 값

array의 마지막 원소 값. 이 값은 배열에서 제거된다.

설명

pop() 메소드는 배열의 마지막 원소를 제거하기 때문에 배열의 length 속성이 1 줄어든다. 또한 이 메소드에서는 배열에서 삭제한 원소를 리턴한다. 배열의 마지막 원소를 제거한다는 점을 제외하면 shift() 메소드와 거의 비슷하다.

예제

```
myList = new Array("one", "two", "three");
trace ("Now deleting " + myList.pop());
// myList는 ["one", "two"]가 된다.
```

참조

Array.push(), Array.shift(), Array.splice(), 11장의 '배열에서 원소 제거하기', 5장의 'delete 연산자'

Array.push() 메소드

배열의 맨 뒤에 하나 이상의 원소를 추가한다.

버전 플래시 5

문법 `array.push(value1, value2, ... valuen)`

인자

value1,...valuen

array의 맨 뒤에 추가할 한 개 이상의 값 리스트

리턴 값

array의 새로운 length 값

설명

push() 메소드에서는 어떤 값의 리스트를 인자로 받아서 그 값들을 배열의 새로운 원소로 추가한다. 각 원소는 인자에 전달된 순서대로 추가된다. 새로운 배열을 만들어내는 concat() 메소드와는 달리 push() 메소드에서는 원본 배열을 수정한다. 또한 unshift()에서는 새로운 원소를 배열의 맨 앞에 추가하지만 push()에서는 맨 뒤에 추가한다.

예제

```
myList = new Array (5, 6);
myList.push(7);           // myList는 [5, 6, 7]이 된다.
myList.push(10, 8, 9);    // myList가 [5, 6, 7, 10, 8, 9]이 된다.
```

참조

`Array.concat()`, `Array.pop()`, `Array.unshift()`, 11장의 ‘배열에 원소 추가하기’

Array.reverse() 메소드

배열에 있는 원소의 순서를 거꾸로 뒤집는다.

버전 플래시 5

문법 `array.reverse()`

설명

`reverse` 메소드는 배열에 있는 원소의 순서를 반대로 뒤집는다. 따라서 마지막 원소가 첫 번째 원소가 되고 마지막에서 두 번째 원소는 두 번째 원소가 된다.

예제

```
myList = new Array(3, 4, 5, 6, 7);
myList.reverse();           // myList가 [7, 6, 5, 4, 3]이 된다.
```

참조

`Array.sort()`

Array.shift() 메소드

배열의 첫 번째 원소를 제거한다.

버전 플래시 5

문법 `array.shift()`

리턴 값

배열에서 제거되는 첫 번째 원소의 값

설명

`shift()` 메소드에서는 배열의 첫 번째 원소를 제거하고 나머지 원소를 모두 한 칸 앞으로 이동시킨다. 따라서 배열의 `length` 속성 값이 1 줄어든다. `pop()` 메소드에서는 배열의 마지막 원소를 제거하고 그 값을 리턴하는 반면에 `shift()`에서는 배열의 첫 번째 원소를 제거하고 그 값을 리턴한다는 점을 기억해 두자.

예제

```
myList = new Array ("a", "b", "c");
myList.shift();           // myList가 ["b", "c"]가 된다.
```

참조

Array.pop(), Array.splice(), Array.unshift(), 11장의 '배열에서 원소 제거하기'

Array.slice() 메소드

원래 있던 배열의 일부를 뽑아내어 새로운 배열을 만든다.

버전 플래시 5

문법 `array.slice(startIndex, endIndex)`

인자

startIndex 새로 만드는 배열에 추가할 첫 번째 원소를 가리키는 인덱스. 이 값이 음수이면 배열의 맨 뒤를 기준으로 거꾸로 숫자를 센다(-1은 마지막 원소, -2는 마지막에서 두 번째 원소 등).

endIndex 새로 만드는 배열에 추가할 마지막 원소 바로 뒤에 있는 원소를 가리키는 인덱스. 이 값이 음수이면 배열의 맨 뒤를 기준으로 거꾸로 숫자를 센다(-1은 마지막 원소, -2는 마지막에서 두 번째 원소 등). 이 값을 생략하면 array.length가 기본값으로 쓰인다.

리턴 값

array 배열의 원소 중에서 startIndex에서 endIndex-1까지 원소가 포함된 새로운 배열

설명

slice() 메소드에서는 원래 있던 배열에서 연속된 원소들을 추출하여 새로운 원소를 만든다. 새로운 배열은 array[startIndex]부터 array[endIndex-1]까지가 포함된 array의 일부분이다.

예제

```
myList = new Array("a", "b", "c", "d", "e");

// myOtherList를 ["b", "c", "d"]로 설정한다.
myOtherList = myList.slice(1, 4);

// anotherList를 ["d", "e"]로 설정한다.
anotherList = myList.slice(3);

// yetAnotherList를 ["c", "d"]로 설정한다.
yetAnotherList = myList.slice(-3, -1);
```

참조

Array.splice(), 11장의 'slice() 메소드', 5장의 'delete 연산자'

Array.sort() 메소드

배열의 원소를 정렬한다.

버전 플래시 5

문법 `array.sort()`
`array.sort(compareFunction)`

인자***compareFunction***

array를 정렬하는 방법을 기술하는 함수

설명

아무런 인자도 전달하지 않으면 sort() 메소드에서는 array의 모든 원소를 임시로 문자열로 변환하고 각 원소를 그 문자열의 코드 포인트(알파벳 순서와 거의 비슷함)를 기준으로 정렬한다. 알파벳 비교와 코드 포인트에 관한 내용은 '4장. 원시 데이터형'의 '문자 순서와 알파벳 순서 비교'에 나와있다.

호출할 때 compareFunction 인자를 전달하면 compareFunction의 리턴 값을 기준으로 배열을 정렬한다. compareFunction은 두 개의 값이 주어졌을 때 그 두 값의 순서를 정해주는 사용자 정의 함수이다. 이 함수를 만들 때는 두 개의 값을 인자로 받아들이도록 만들어야 한다. 이 함수에서는 만약 첫 번째 인자가 두 번째 인자

보다 앞으로 가야 한다면 음수를, 뒤로 가야 한다면 양수를, 순서를 그대로 뒤야 한다면 0을 리턴한다. 한 번 정렬한 배열에 새로운 원소를 추가한다고 해서 새로 추가되는 원소도 정렬된 순서로 들어가는 것은 아니다. 새로운 원소를 추가하고 나서도 배열이 정렬된 상태를 유지하도록 하려면 원소를 추가한 뒤에 다시 `sort()` 함수를 호출해야 한다. 정렬을 할 때는 Latin 1 코드를 기준으로 정렬하는 것이 기본으로 설정되어 있다. 숫자 값의 크기에 따라 숫자를 정렬하는 방법은 11장에 나와 있다.

예제

아래 예제에서는 무비 클립의 배열을 스크린에서의 수평 위치를 기준으로 정렬한다.

```
var clips = [clip1, clip2, clip3, clip4];

function compareXposition (element1, element2) {
  if (element1._x < element2._x) {
    return -1;
  } else if (element1._x > element2._x) {
    return 1;
  } else {
    return 0; // 두 클립의 수평 위치가 똑같은 경우
  }
}
```

참조

`Array.reverse()`, 11장의 ‘`sort()` 메소드’

Array.splice() 메소드

배열에서 원소를 제거하고 새로운 원소를 추가한다.

버전 플래시 5

문법

```
array.splice(startIndex)
array.splice(startIndex, deleteCount)
array.splice(startIndex, deleteCount,
              value1,...,valuen)
```

인자

startIndex 원소를 지우고 새로운 원소를 추가(옵션)하기 시작하는 위치의 인덱스. 이 값이 음수이면 배열의 맨 뒤를 기준으로 거꾸로 센다(-1은 마지막 원소, -2는 마지막에서 두 번째 원소 등).

deleteCount

배열에서 제거할 원소의 개수를 나타내는 0 이상의 정수(옵션). *startIndex* 위치에 있는 원소의 개수도 포함된다. 이 값을 0으로 지정하면 아무 원소도 제거되지 않는다. 이 값을 생략하면 *startIndex*에서 배열의 끝까지 모든 원소를 삭제한다.

value1,...,valueN

원소를 제거한 뒤에 *startIndex* 위치에 추가할 값들의 리스트(옵션)

리턴 값

제거된 원소가 들어있는 새 배열이 리턴된다(원본 배열은 함수를 호출할 때 지정한 방법대로 바뀐다).

설명

`splice()` 메소드를 이용하면 `array[startIndex]`부터 `array[startIndex + deleteCount - 1]`까지의 원소를 삭제할 수 있고 *startIndex* 위치에 새로운 원소들을 추가할 수도 있다. `splice()` 메소드를 이용해도 배열에 빈 자리가 생기지는 않는다. 제거된 부분 뒤에 있는 원소들을 앞으로 옮겨서 빈 자리를 모두 메꾸기 때문이다.

예제

```
myList = new Array (1, 2, 3, 4, 5);
// 배열의 두 번째와 세 번째 원소를 제거하고 그 자리에
// "x", "y", "z"를 삽입한다.
// 이렇게 하면 myList는 [1, "x", "y", "z", 4, 5]가 된다.
myList.splice(1, 2, "x", "y", "z");
```

참조

`Array.slice()`, 11장의 ‘`splice()` 메소드’

Array.toString() 메소드

배열의 각 원소를 쉼표로 구분하여 하나의 문자열로 만든다.

버전 플래시 5

문법 `array.toString()`

리턴 값

array의 각 원소를 쉼표로 구분한 문자열

설명

toString() 메소드에서는 array를 문자열로 변환한 값을 리턴한다. toString()에서 리턴한 문자열은 각 원소를 문자열로 변환하고 그 문자열들을 쉼표로 구분하여 합친 값이다(인자 없이 join() 메소드를 호출하는 것과 똑같다). 문자열이 들어갈 위치에 배열을 사용하면 자동으로 toString() 메소드가 호출된다. 따라서 toString() 메소드를 직접 호출하는 경우는 거의 없다. 일반적으로 어떤 배열을 문자열로 바꿀 때는 문자열의 형식을 자유롭게 조절할 수 있는 join() 메소드를 사용한다.

예제

```
myList = new Array("a", "b", "c");           // 배열을 만든다.
trace(myList.toString());                   // "a","b","c"가 출력된다.
myList = new Array([1, 2, 3], "a", "b", "c"); // 중첩된 배열을 만든다.
trace(myList.toString());                   // "1,2,3,a,b,c"가 출력된다.
```

참조

Array.join()

Array.unshift() 메소드

배열의 맨 앞에 한 개 이상의 원소를 추가한다.

버전 플래시 5

문법 `array.unshift(value1, value2, ... valuen)`

인자

value1...valuen

array의 맨 앞에 추가할 한 개 이상의 원소 리스트

리턴 값

array의 새로운 length 값

설명

unshift() 메소드에서는 배열의 맨 앞에 일련의 원소를 추가한다. 각 원소는 인자에 입력한 순서대로 추가된다. 배열의 맨 뒤에 원소를 추가할 때는 push()를 사용한다.

예제

```
myList = new Array (5, 6);  
myList.unshift(4);           // myList는 [4, 5, 6]이 된다  
myList.unshift(7, 1);        // myList는 [7, 1, 4, 5, 6]이 된다.
```

참조

Array.push(), Array.shift(), 11장의 '배열에 원소 추가하기'

Boolean() 전역 함수

주어진 값을 부울 데이터형으로 변환한다.

버전 플래시 5

문법 Boolean(value)

인자

value 부울형으로 변환될 값을 나타내는 표현식

리턴 값

value를 부울형으로 변환한 값(true 또는 false)

설명

Boolean() 전역 함수는 주어진 인자를 부울 값으로 변환하여 그 값을 리턴한다. 부울형이 아닌 데이터를 부울형으로 변환할 때는 [표 3-3]에 나온 규칙을 따른다. 이 함수는 그다지 자주 쓰이는 함수는 아니다. 액션스크립트에서 필요에 따라 주어진 값을 자동으로 부울형으로 변환해주기 때문이다.

주의 사항

Boolean() 전역 함수와 Boolean 클래스 생성자를 혼동하지 않도록 주의하자. Boolean 함수는 주어진 인자를 부울 데이터형으로 변환하는 함수이지만, Boolean 클래스 생성자는 새로운 부울형 객체를 만드는 생성자이다. ECMA-262에서는 비어 있지 않은 문자열은 모두 true로 변환하도록 되어 있지만, 플래시 5에서는 숫자(0 제외)로 변환할 수 있는 문자열만 true가 된다. 따라서 "true"라는 문자열도 부울형으로 변환하면 false가 된다.

예제

```
var x = 1;
if (Boolean(x)) {
    trace("x is true");
}
```

참조

Boolean 클래스, '3장. 데이터 및 데이터형'의 '데이터형 변환'

Boolean 클래스

원시 부울형 데이터의 래퍼 클래스

버전 플래시 5

생성자 new Boolean(value)

인자

value Boolean 생성자에서는 이 표현식 값을 계산하고 부울형으로 변환하여 Boolean 객체를 만든다.

메소드

toString() Boolean 객체의 값을 문자열로 변환하는 메소드

valueOf() Boolean 객체의 값을 원시 데이터값으로 변환하는 메소드

설명

Boolean 클래스에서는 외부에서 액세스할 수 없는 내부 속성에 원시 부울 값을 저장해 두는 Boolean 객체를 만든다. Boolean 객체는 Boolean 클래스의 메소드를 이용하여 원시 부울 값을 검사하고 조작하기 위한 용도로만 쓰인다. Boolean 객체는 래퍼 객체(wrapper object)의 일종이다. 원시 부울 값을 감싸서 다른 객체에 있는 메소드를 사용할 수 있게 하는 용도로만 쓰이기 때문이다. 문자열이나 숫자를 감싸기 위한 래퍼 클래스인 String이나 Number 클래스와 비교해 보자(Boolean 클래스에 비해 String이나 Number 클래스는 더 다양한 기능을 제공한다).

대부분의 경우에 Boolean 객체는 액션스크립트 내부에서만 쓰인다. 원시 부울 값에 대해 어떤 메소드를 호출할 때마다 인터프리터에 의해 자동으로 생성되며, 작업이 끝나고 나면 자동으로 삭제된다. Boolean 생성자를 이용하여 Boolean 객체를 만들 수도 있지만 굳이 Boolean 객체를 만들어서 쓸 일이 거의 없다.

주의 사항

데이터형을 변환할 때는 대부분 Boolean 클래스보다는 Boolean() 전역 함수를 사용한다.

참조

Boolean() 전역 함수, 4장의 '부울형'

Boolean.toString() 메소드

Boolean 객체의 값을 문자열로 변환한 값

버전 플래시 5

문법 *booleanObject.toString()*

리턴 값

booleanObject의 원시 값이 true이면 "true"를, false이면 "false"를 리턴한다. Boolean 객체의 값은 객체를 만들 때 지정해야 하며 그 후에는 내부적으로 저장된다. Boolean 객체 내부에 저장된 속성을 직접 사용할 수는 없지만, toString()을 이용하면 그 값을 문자열로 변환한 값을 알아낼 수 있다.

설명

toString() 메소드에서는 Boolean() 객체의 원시 값을 구해서 그 값을 문자열로 변환하여 그 문자열을 리턴한다.

예제

```
x = new Boolean(true);
trace(x.toString()); // "true"가 출력된다.
```

참조

Object.toString()

Boolean.valueOf() 메소드

Boolean 객체의 원시 값

버전 플래시 5

문법 *booleanObject*.valueOf()

리턴 값

booleanObject의 원시 값이 true이면 부울 값 true를, false이면 부울 값 false를 리턴한다. Boolean 객체의 값은 객체를 만들 때 지정해야 하며, 그 후에는 내부적으로 저장된다.

설명

valueOf() 메소드에서는 Boolean 객체에 해당하는 원시 부울 데이터를 리턴한다. Boolean 객체에 저장된 속성을 직접 사용할 수는 없지만, valueOf() 메소드를 이용하여 그 객체에 해당하는 부울 값을 구할 수 있다.

예제

```
x = new Boolean(0);
trace(x.valueOf()); // false가 출력된다.
```

참조

Object.valueOf()

call() 전역 함수

원격 프레임의 스크립트를 실행

버전 플래시 4(플래시 5에서는 더 이상 쓰이지 않는다)**문법**
`call(frameLabel)`
`call(frameNumber)`**인자*****frameLabel***

실행할 스크립트가 들어있는 프레임을 나타내는 문자열 레이블

frameNumber

스크립트를 실행할 프레임 번호

설명

`call()` 함수는 `frameLabel`이나 `frameNumber`가 가리키는 프레임에 들어있는 스크립트를 실행시킨다. 예를 들어 다음과 같은 코드를 이용하면 현재 타임라인의 20번 프레임에 있는 코드를 실행시킬 수 있다.

```
call(20);
```

플래시 4에서는 이 함수로 재사용할 수 있는 서브루틴(물론 인자를 받아들이거나 어떤 값을 리턴하지는 못한다)의 기능을 대신하지만, 플래시 5에서는 `function` 선언문을 사용하는 것이 바람직하다.

플래시 5에서 `call()` 함수를 이용하여 원격 함수를 호출하면 그 부분에서 `var` 키워드를 이용하여 선언한 모든 변수는 지역 변수가 되기 때문에, 스크립트가 완료되면 자동으로 없어진다. 원격 실행되는 스크립트에서 만든 변수를 계속해서 사용해야 한다면 `var` 키워드를 빼서 타임라인 변수로 만들면 된다.

```
var x = 10; // 지역 변수이므로 스크립트가 끝나고 나면 없어진다.  
x = 10;    // 타임라인 변수가 되어 스크립트가 끝나도 그대로 남는다.
```

다른 타임라인에 있는 프레임에 대해 `call()` 함수를 사용하려면 `tellTarget()` 함수를 이용하면 된다. 다음과 같은 코드를 이용하면 `box` 클립의 10번 프레임에 있는 스크립트를 실행시킬 수 있다.

```
tellTarget ("box") {
    call(10);
}
```

참조

'9장. 함수', '부록 C. 하위 호환성'

Color 클래스

무비 클립 색 조절

버전 플래시 5

생성자 `new Color(target)`

인자

target 새로 만든 객체로 색을 조절할 무비 클립이나 문서 레벨의 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 자동 변환된다)

메소드

getRGB() 빨간색(R), 녹색(G), 파란색(B)의 오프셋 값을 구하는 함수

getTransform()

빨간색, 녹색, 파란색, 알파의 오프셋과 백분율 값을 구하는 함수

setRGB() 빨간색, 녹색, 파란색의 오프셋 값을 새로 설정한다. 백분율 값은 0으로 만든다.

setTransform()

빨간색, 녹색, 파란색, 알파의 새로운 오프셋 및 백분율 값을 설정한다.

설명

Color 클래스의 객체를 이용하면 프로그래밍을 통해 무비 클립이나 메인 무비의 색과 투명도를 조절할 수 있다. 어떤 대상에 대한 Color 클래스의 객체를 만들고 나면 그 객체의 메소드를 호출하여 대상 객체의 색과 투명도를 바꿀 수 있다. 예를 들

어 ball이라는 클립 인스턴스의 색을 빨간색으로 바꾸는 경우를 생각해 보자. 우선 ball 인스턴스를 인자로 전달하여 Color 객체를 만들고 그 객체를 ballColor라는 변수에 저장한다. 그리고 나서 ballColor.setRGB()를 호출하여 ball의 색을 빨간색으로 바꾼다.

```
var ballColor = new Color("ball");
ballColor.setRGB(0xFF0000);
// setRGB()에 빨간색을 나타내는 16진수 값을 전달한다.
```

간단한 애플리케이션에서는 위와 같은 방법만으로도 적절히 색을 조절할 수 있다. 하지만 더 다양한 방법으로 색을 조절하기 위해서는 플래시에서 색을 표현하는 방법을 확실히 이해해야 한다. 무비 클립에 나타나는 모든 색은 빨간색(R), 녹색(G), 파란색(B), 알파(Alpha, 투명도라고도 부름)라는 네 개의 서로 다른 성분으로 이루어진다. 이 네 가지 성분을 조합하면 화면에 표시되는 거의 모든 색을 만들 수 있다. 빨간색, 녹색, 파란색, 투명도는 각각 0에서 255까지의 숫자로 표현된다. 숫자가 클수록 그 숫자에 해당하는 색의 영향이 커진다. 하지만 컴퓨터에서 사용하는 색은 그림을 그릴 때와는 달리 색을 더할수록 어두워지지 않고 더 밝아진다. RGB의 세 가지 성분의 값이 같으면 회색이 되며, 세 값이 모두 0이면 검은색, 모두 255이면 흰색이 된다. 알파 값이 커지면 색이 불투명해진다(알파 값이 0인 색은 완전히 투명한 색이 되고 알파 값이 255이면 완전히 불투명한 색이 된다).

예를 들어 순수한 빨간색은 다음과 같은 값으로 표현할 수 있다.

```
Red: 255, Green: 0, Blue: 0, Alpha: 255
```

약간 투명한 빨간색은 다음과 같은 값으로 나타낼 수 있다.

```
Red: 255, Green: 0, Blue: 0, Alpha: 130
```

지금은 우선 편의상 RGB(빨간색, 녹색, 파란색) 값을 이용하여 색을 표기하겠다. 액션스크립트에서는 (255, 0, 255)와 같은 식으로 색을 표기할 수 없다. 대신 0xRRGGBB와 같은 형태의 16진수 표기법을 사용하는데, RR, GG, BB는 각각 빨간색, 녹색, 파란색을 나타내는 두 자리 16진수 값이다. 또한 RGBA(빨간색, 녹색, 파란색, 알파) 표기법도 사용하겠다(이러한 표기법은 실제 액션스크립트에서 쓰이는 것은 아니고 여기서 편의상 도입한 표기법일 뿐이다).

무비 클립에 있는 각 색깔의 빨간색, 녹색, 파란색, 알파 값은 플래시 저작도구에 있는 Mixer 패널을 통해 설정한다(Mixer 패널에서 알파 값은 0부터 255까지의 값이 아니라 백분율로 표기된다). 액션스크립트를 이용하여 무비 클립의 모든 색깔을 한꺼번에 바꾸고 싶다면 클립 색깔의 빨간색, 녹색, 파란색, 알파 성분을 한꺼번에 수정하면 된다(이러한 작업을 변환이라고 부른다).

다음과 같은 두 가지 방법으로 각 색 성분의 변환 방법을 설정할 수 있다.

- 성분의 원래 값의 백분율을 -100에서 100 사이의 값으로 설정할 수 있다. 이러한 방법으로 클립에 있는 모든 빨간색을 원래 값의 80%로 설정할 수 있다.
- 성분의 원래 값에 오프셋을 지정할 수도 있다. 이 때 오프셋 값으로 -255에서 255까지의 정수를 사용할 수 있다. 예를 들어 클립에 있는 모든 파란색 값에 20을 더하거나 음수를 사용하여 클립에 있는 모든 파란색 값에서 30을 빼는 것도 가능하다.

변환된 클립의 최종 색깔은 원본의 색 성분과 Color 객체를 이용하여 설정한 백분율 및 오프셋 값에 의해 결정된다.

```
R = originalRedValue * (redTransformPercentage/100)
+ redTransformOffset
G = originalGreenValue * (greenTransformPercentage/100)
+ greenTransformOffset
B = originalBlueValue * (blueTransformPercentage/100)
+ blueTransformOffset
A = originalAlphaValue * (alphaTransformPercentage/100)
+ alphaTransformOffset
```

액션스크립트를 통해 변환하면 백분율의 초기 값은 100으로, 오프셋의 초기 값은 0으로 설정된다.

몇 가지 예를 통해 색 변환이 어떤 식으로 처리되는지 살펴보자. 어떤 클립에 불투명한 빨간색 삼각형(R:255, G:0, B:0, A:255)과 불투명한 녹색 원(R:0, G:255, B:0, A:255)이 있다고 가정하자. 이제 녹색의 백분율을 50으로, 알파의 백분율을 80으로, 파란색의 오프셋을 100으로 설정하고 나머지 백분율 및 오프셋은 기본값을 그대로 사용하여 클립 전체의 색을 변환시켜보자. 이러한 작업을 거치면 빨간색 삼각형은 다음과 같이 변환된다.

```

R == 255 * (100/100) + 0 == 255      // 빨간색 값은 바뀌지 않는다.
G == 0 * (50/100) + 0 == 0           // 녹색 값은 50%로 줄어든다.
B == 0 * (100/100) + 100 == 100      // 파란색 값은 100 커진다.
A == 255 * (80/100) + 0 == 204      // 알파 값은 80%로 줄어든다.

```

따라서 변환된 빨간 삼각형은 (R:255, G:0, B:100, A:204) 값을 가지게 된다. 녹색 원은 다음과 같이 변환된다.

```

R == 0 * (100/100) + 0 == 0          // 빨간색 값은 바뀌지 않는다.
G == 255 * (50/100) + 0 == 127.5    // 녹색 값은 50%로 줄어든다.
B == 0 * (100/100) + 100 == 100     // 파란색 값은 100 커진다.
A == 255 * (80/100) + 0 == 204     // 알파 값은 80%로 줄어든다.

```

녹색 원은 (R:0, G:127.5, B:100, A:204)로 변환된다.

위에서 예로 들었던 색 변환을 실제 클립에 적용할 때는 Color 객체를 이용한다. 클립의 전체 색의 오프셋과 백분율 값을 설정할 때는 setRGB()나 setTransform() 메소드를 사용한다(각 메소드에 해당하는 예제 참고). 어떤 클립의 색 변환 상태를 보고 싶다면 getRGB()와 getTransform() 메소드를 이용하면 된다. Color 클래스 메소드를 이용하면 페이드인이나 페이드아웃, 색을 흐리게 만드는 것과 같은 동적인 색 효과를 구현할 수 있다. 또한 Color 클래스를 이용하면 클립 인스턴스마다 색 변환을 적용할 수 있기 때문에, 최소한의 자원으로 다양한 그래픽 효과를 얻을 수 있다. 예를 들어 아래 예제에 나온 것처럼 하나의 무비 클립의 색을 다양한 방법으로 변환하여 풍선으로 가득한 장면을 만들어낼 수도 있다.

주의 사항

Color 객체를 사용할 때는 다음과 같은 점에 주의해야 한다.

- Color 객체를 이용하여 무비 클립의 색을 바꾸면 무비 클립이 액션스크립트에 의해서만 제어되므로 무비를 만들 때 클립에 적용했던 기능들이 더 이상 작동하지 않을 수도 있다.
- 클립의 _alpha 속성 값을 새로 설정하면 클립의 알파 백분율 값이 바뀌며, 이러한 내용은 getTransform()에서 리턴한 객체의 aa라는 속성에서 확인할 수 있다.

- 색을 변환한다고 해도 무비나 무비 클립의 배경색은 바뀌지 않는다. 색 변환은 스테이지에 있는 단색 도형에만 적용된다.
- Effect 패널(Window → Panels → Effect)을 통해 저작 도구에서 수동으로 무비 클립 색을 변환할 수 있다. 이렇게 색을 변환한 결과는 `getTransform()`에서 리턴된 객체의 속성을 통해 확인할 수 있다. Effect 패널은 무비를 만들 때 색 변환 상태를 확인하고 설정하는 데 매우 유용하다.

예제

첫 번째 예제에서는 balloon(풍선)이라는 클립을 기반으로 무작위로 색을 입힌 여러 개의 풍선을 만드는 방법을 보여준다.

```
// 루프를 통해 balloon 클립의 복사본을 20개 만든다.
for (var i = 0; i < 20; i++) {
    // balloon 클립을 복사한다.
    balloon.duplicateMovieClip("balloon" + i, i);

    // 복사본을 스테이지 위에 놓는다.
    this["balloon" + i]._x = Math.floor(Math.random() * 550);
    this["balloon" + i]._y = Math.floor(Math.random() * 400);

    // 이 풍선에 적용할 Color 객체를 만든다.
    balloonColor = new Color(this["balloon" + i]);
    // setRGB() 메소드를 이용하여 풍선의 색을 임의로 설정한다.
    balloonColor.setRGB(Math.floor(Math.random() * 0xFFFFFF));
}
```

빨간색, 녹색, 파란색 오프셋을 같은 값으로 설정하면 무비 클립을 밝게 또는 어둡게 하는 작업을 간단히 처리할 수 있다. 다음과 같은 코드를 사용하면 `myClip`을 어둡게 만들 수 있다.

```
brightness = new Color("myClip");
brightnessTransform = new Object();
brightnessTransform.rb = -30;
brightnessTransform.bb = -30;
brightnessTransform.gb = -30;
brightness.setTransform(brightnessTransform);
```

마지막 예제는 마우스 위치에 따라 클립을 밝게 또는 어둡게 만드는 코드이다.

```
onClipEvent (load) {
    var brightness = new Color(this);
    var brightnessTransform = new Object();
    var stageWidth = 550;
}

onClipEvent (mouseMove) {
    brightnessAmount = -255 + (_root._xmouse / stageWidth) * 510;
    brightnessTransform.rb = brightnessAmount;
    brightnessTransform.bb = brightnessAmount;
    brightnessTransform.gb = brightnessAmount;
    brightness.setTransform(brightnessTransform);
    updateAfterEvent();
}
```

Color.getRGB() 메소드

빨간색, 녹색, 파란색의 현재 오프셋 값을 구하는 메소드

버전 플래시 5

문법 `colorObj.getRGB()`

리턴 값

colorObj의 대상 객체의 현재 RGB 오프셋을 나타내는 숫자

설명

getRGB() 메소드에서는 클립의 빨간색, 녹색, 파란색 성분의 오프셋 값을 나타내는 -16777215에서 16777215까지의 정수를 리턴한다. 색 백분율을 구하려면 getTransform() 메소드를 이용해야 한다. 색 오프셋 값은 0에서 255까지의 값으로 표현되므로 각 색의 오프셋이 두 자리 16진수로 표현될 수 있도록 getRGB()의 리턴 값을 16진수로 확인하는 것이 좋다. 즉 getRGB()에서 리턴받은 숫자를 볼 때는 그 숫자를 0xRRGGBB(RR: 빨간색, GG: 녹색, BB: 파란색) 형태의 여섯 자리 16진수로 만드는 것이 좋다. 예를 들어 getRGB()에서 10092339라는 값이 리턴되었다면 그 숫자를 16진수로 변환하면 0x99FF33이 되므로 오프셋은 R:153, G:255, B:51이 된다는 것을 알 수 있다. 이와 마찬가지로 리턴 값이 255라면 16진수로 변

환한 값은 0x0000FF가 되어 오프셋이 R:0, G:0, B:255가 된다는 것을 알 수 있다. getRGB()에서 리턴한 값을 16진수로 변환할 때는 다음과 같이 toString() 메소드를 이용하면 된다.

```
// Color 객체를 만든다.
myColor = new Color("myClip");
// 빨간색 오프셋을 255(16진수로는 FF)로 설정한다.
myColor.setRGB(0xFF0000);
// RGB 오프셋을 구해서 16진수로 변환한다.
hexColor = myColor.getRGB().toString(16);
trace(hexColor); // ff0000가 출력된다.
```

HTML 태그에서 보통 16진수로 색을 표현하기 때문에 웹 개발자라면 이러한 표기법에 익숙할 것이다. 예를 들어 어떤 HTML 페이지에서 배경색을 설정할 때는 다음과 같이 16진수 색 표기법을 사용한다(다음과 같은 코드를 이용하면 배경색은 빨간색과 파란색을 합친 분홍색이 된다).

```
<BODY BGCOLOR="#FF00FF">
```

HTML 태그에서 사용하는 16진수 표기법은 getRGB()와 setRGB()에서 사용하는 형식과 똑같다. 하지만 getRGB()의 리턴 값을 반드시 16진수로 써야 하는 것은 아니다. 비트 연산자를 이용하여 getRGB()의 리턴 값으로부터 빨간색, 녹색, 파란색 오프셋을 각각 추출할 수도 있다.

```
var rgb = myColorObject.getRGB();
var red = (rgb >> 16) & 0xFF;
// 빨간색 오프셋만 뽑아내서 red에 그 값을 대입한다.
var green = (rgb >> 8) & 0xFF;
// 녹색 오프셋만 뽑아내서 green에 그 값을 대입한다.
var blue = rgb & 0xFF;
// 파란색 오프셋만 뽑아내서 blue에 그 값을 대입한다.
```

이렇게 각 색깔의 오프셋 값을 분리해 놓으면 각 오프셋 값을 10진수로 알아낼 수도 있고, 그 값을 조작할 수도 있다. 하지만 Color 객체의 오프셋 값을 변경할 때는 setRGB() 메소드를 설명하는 부분에 나온 것처럼 각 오프셋을 모두 합쳐서 하나의 숫자로 다시 바꿔줘야 한다.

주의 사항

클립의 원래 색을 참조하지 않고 바로 새로운 색을 설정할 때는 `getRGB()`와 `setRGB()` 메소드를 사용하는 것이 좋다. 하지만 클립의 원래 색을 기준으로 색 변환을 수정할 때는 `getTransformation()`과 `setTransformation()` 메소드를 사용하는 것이 낫다.

색 오프셋을 구할 때는 `getTransform()`을 이용하는 것이 가장 편리하다. `getRGB()`와 같이 하나의 숫자로 모든 값을 한꺼번에 리턴하는 것이 아니라 각 성분별로 오프셋 값을 따로 알 수 있기 때문이다. 특히 음수를 이진수나 16진수로 표현하기 까다롭기 때문에, 오프셋을 음수로 설정하는 경우에는 `setTransform()` 메소드를 사용하는 것이 훨씬 편하다.

예제

```
// box라는 클립에 대해 새로운 Color 객체를 만든다.
boxColor = new Color("box");
// box에 새로운 RGB 오프셋을 설정한다.
boxColor.setRGB(0x333366);
// box의 RGB 오프셋을 확인한다.
trace(boxColor.getRGB()); // 3355494가 출력된다.
```

참조

`Color.getTransform()`, `Color.setRGB()`

Color.getTransform() 메소드

클립의 빨간색, 녹색, 파란색, 알파 성분의 오프셋 및 백분율 값을 구하는 메소드

버전 플래시 5

문법 `colorObj.getTransform()`

리턴 값

`colorObj`의 대상이 되는 클립의 색 변환 값들이 속성으로 저장된 객체

설명

getTransform() 메소드에서는 Color 객체의 대상이 되는 클립에 적용된 색 변환 내역을 속성으로 저장하고 있는 객체를 리턴한다. 리턴된 객체의 속성 이름과 속성 값은 [표 R-4]에 나와 있다.

[표 R-4] getTransform()에서 리턴하는 객체의 속성

속성 이름	속성 값	설명
ra	-100 ~ 100	빨간색 변환 비율
rb	-255 ~ 255	빨간색 오프셋
ga	-100 ~ 100	녹색 변환 비율
gb	-255 ~ 255	녹색 오프셋
ba	-100 ~ 100	파란색 변환 비율
bb	-255 ~ 255	파란색 오프셋
aa	-100 ~ 100	알파 변환 비율
ab	-255 ~ 255	알파 오프셋

주의 사항

[표 R-4]를 보면 오프셋과 백분율 값 모두 음수가 될 수 있다. 하지만 이러한 음수 값은 RGB 성분을 계산하는 과정에서만 쓰이며, 각 성분을 계산한 값은 0 이상, 255 이하가 되어야 한다. 이 범위를 벗어나는 값들은 자동으로 이 범위 안에 들어오도록 조절된다. RGB와 알파 성분 값을 계산하는 방법은 Color 클래스를 설명하는 부분에 나와 있다.

예제

getTransform()과 setTransform()을 활용하면 색 변환에서 빨간색, 녹색, 파란색, 알파 성분을 각각 따로 설정할 수 있다. 아래 코드는 box라는 클립의 빨간색과 알파 성분만 조절하는 예제이다.

```
// box라는 클립을 대상으로 하는 Color 객체를 새로 만든다.
boxColor = new Color("box");

// getTransform() 함수에서 리턴한 객체를 boxTransform에 대입한다.
boxTransform = boxColor.getTransform();

// 변환 객체의 속성을 조절한다.
```

```

boxTransform.rb = 200;      // 빨간색 오프셋을 200으로 설정한다.
boxTransform.aa = 60;      // 알파 비율을 60%로 설정한다.

// boxColor 객체를 통해서 새로운 변환을 적용한다.
boxColor.setTransform(boxTransform);

```

참조

Color.setTransform()

Color.setRGB() 메소드

빨간색, 녹색, 파란색의 새로운 오프셋 값을
대입하는 메소드

버전 플래시 5

문법 `colorObj.setRGB(offset);`

인자

offset colorObj의 대상 객체에 새로 설정할 RGB 오프셋을 나타내는 0 이상 16777215 (0xFFFFFFFF) 이하의 정수. 10진수나 16진수를 모두 사용할 수 있다.

허용된 범위를 벗어나는 숫자는 자동으로 허용된 범위 이내의 숫자로 변환된다(2의 보수 이진수 표기법을 적용). 따라서 setRGB()에서는 setTransform()에서와 같이 음수 오프셋을 사용할 수 없다.

설명

setRGB() 메소드에서는 무비 클립의 RGB 성분에 대한 새로운 변환 값을 대입하는 데 쓰인다. 새로운 오프셋은 0xRRGGBB 형태의 여섯 자리 16진수로 보통 표기되는데, 이 때 RR, GG, BB는 각각 빨간색, 녹색, 파란색 성분을 나타내는 00에서 FF까지의 두 자리 16진수이다. 예를 들어 (51, 51, 102)라는 RGB 쌍은 다음과 같은 16진수로 표기한다.

0x333366

menu라는 클립에 회색 오프셋을 대입하려면 다음과 같은 코드를 사용하면 된다.

```
var menuColor = new Color("menu");
menuColor.setRGB(0x999999);
```

HTML에서 색을 16진수로 표현하는 방법에 익숙한 웹 개발자라면 setRGB()를 호출할 때 위와 같이 16진수 형태로 색을 표현하는 것이 그다지 생소하지 않을 것이다. 10진수, 8진수, 16진수에 대한 기초적인 내용은 <http://www.moock.org/asdg/technotes>에서 찾아볼 수 있다.

setRGB()를 이용하면 클립의 색 변환에서 빨간색, 녹색, 파란색의 백분율이 자동으로 0으로 설정된다. 따라서 setRGB()를 이용하여 색을 바꾸면 클립의 원래 색을 수정하는 것이 아니라 직접 색을 대입하게 된다. 클립의 원래 색을 기준으로 색을 조절할 때는 setTransform() 메소드를 사용한다.

예제

아래 코드에서는 setRGB() 메소드를 사용할 때 인자로 전달할 숫자를 쉽게 만들 수 있는 방법을 보여준다. 아래 코드에 나와 있는 combineRGB() 함수에서는 red와 green 값을 시프트시킨 다음, 비트 OR 연산자를 이용하여 blue와 합쳐서 24비트로 이루어진 숫자를 만든다. 이렇게 만든 숫자를 이용하여 box라는 무비 클립의 색을 설정할 수 있다.

```
function combineRGB (red, green, blue) {
    // 세 개의 색상 값을 하나의 숫자로 합친다.
    var RGB = (red<<16) | (green<<8) | blue;
    return RGB;
}
// Color 객체를 만든다.
var boxColor = new Color("box");
// box 클립의 RGB 값을 (201, 160, 21)로 설정한다.
boxColor.setRGB(combineRGB(201, 160, 21));
```

비트 단위 연산에 대한 자세한 내용은 '15장. 고급 주제'에 나온다.

참조

Color.getRGB(), Color.setTransform()

Color.setTransform() 메소드

빨간색, 녹색, 파란색, 알파의 오프셋과
백분율 값을 설정하는 메소드

버전 플래시 5

문법 `colorObj.setTransform(transformObject)`

인자

transformObject

colorObj의 대상 클립에 적용할 색 변환 값이 속성으로 저장된 객체

설명

setTransform() 메소드를 이용하면 무비 클립의 색을 빨간색, 녹색, 파란색, 알파 성분별로 오프셋과 백분율 값을 설정하여 세밀하게 조절할 수 있다. setTransform() 메소드를 사용하려면 우선 몇 가지 미리 정의된 속성을 가지는 객체를 만들어야 한다. Color 객체에 적용되는 변환 방법은 이러한 객체의 속성으로 표현되며 각 속성에 대한 설명은 getTransform() 메소드를 설명하는 부분에 있는 [표 R-4]에 나와 있다.

[표 R-4]에 나온 것과 같은 속성을 가지고 있는 객체를 만들고 나면 그 객체를 setTransform() 메소드에 전달한다. transformObject에 저장된 속성 값은 colorObj의 새로운 오프셋 및 백분율 변환 값이 되어 colorObj의 대상 무비 클립의 색이 바뀌게 된다. 최종적인 색상은 Color 클래스를 설명하는 부분에 나온 규칙에 의해 결정된다.

어떤 Color 객체의 오프셋과 백분율 값을 확인할 때는 getTransform() 메소드를 이용하면 된다.

예제

```
// box 클립에 적용할 새로운 Color 객체를 만든다.
boxColor = new Color("box");

// 새로운 익명 객체를 만들어서 boxTransform에 저장한다.
boxTransform = new Object();

// boxTransform에 필요한 속성을 대입하여
// 원하는 변환 값을 입력한다.
```

```

boxTransform.ra = 50;    // 빨간색 백분율
boxTransform.rb = 0;    // 빨간색 오프셋
boxTransform.ga = 100;  // 녹색 백분율
boxTransform.gb = 25;   // 녹색 오프셋
boxTransform.ba = 100;  // 파란색 백분율
boxTransform.bb = 0;    // 파란색 오프셋
boxTransform.aa = 40;   // 알파 백분율
boxTransform.ab = 0;    // 알파 오프셋

// 이렇게 준비한 변환 객체를 setTransform()에 전달한다.
boxColor.setTransform(boxTransform);

```

위와 같은 방법으로 변환 객체를 직접 만들려면 꽤 복잡하다. 다음과 같이 `getTransform()` 메소드를 활용하면 새로운 변환 객체를 조금 더 간단하게 만들 수 있다.

```

// box 클립을 대상으로 하는 새로운 Color 객체를 만든다.
boxColor = new Color("box");

// boxColor에서 getTransform() 메소드를 호출하여
// 변환 객체를 가져온다.
boxTransform = boxColor.getTransform();

// boxTransform 객체에서 필요한 부분만 수정한다.
boxTransform.rb = 51;    // 빨간색 오프셋
boxTransform.aa = 40;    // 알파 백분율

// setTransform() 메소드에 변환 객체를 전달한다.
boxColor.setTransform(boxTransform);

```

참조

`Color.getTransform()`

Date() 전역 함수

날짜와 시각을 나타내는 문자열

버전 플래시 5

문법 `Date()`

리턴 값

현재 날짜와 시각을 나타내는 문자열

설명

Date() 함수에서는 지역 시간대를 기준으로 하는 현재 날짜와 시각을 나타내는 문자열을 리턴한다. 이 문자열에는 GMT 오프셋(지역 시간대와 그리니치 표준시와의 차이)도 함께 기록된다.

Date() 전역 함수와 Date() 클래스 생성자를 혼동하지 않도록 주의하자. Date() 함수에서는 표준 형식을 따르는 간결하게 표현된 날짜와 시각을 리턴할 뿐이다. 일반적으로 날짜와 시각을 보기에는 좋지만 프로그램에서 날짜와 시각을 이용하여 다른 작업을 할 때는 그다지 유용하지 않다. 따라서 그런 경우에는 연, 월, 일, 시각을 따로 사용하기 좋은 Date 클래스의 객체를 이용하는 것이 좋다.

예제

텍스트 필드에 현재 날짜와 시각을 표시하는 가장 쉬운 방법은 다음과 같이 Date() 함수를 이용하는 방법이다.

```
myTextField = Date();

// myTextField에 다음과 같은 형식의 문자열이 출력된다.
// "Mon Aug 28 16:23:09 GMT-0400 2000"
```

참조

Date 클래스, Date.UTC()

Date 클래스

현재 시각 및 날짜 정보

버전 플래시 5

생성자 `new Date()`
 `new Date(milliseconds)`
 `new Date(year, month, day, hours, minutes, seconds, ms)`

인자***milliseconds***

1970년 1월 1일 0시(UTC: 세계 협정시, GMT와 유사함)와 새로운 날짜 사이의 시간을 밀리초(1/1000초) 단위로 센 숫자. 1970년 1월 1일 0시 이전은 음수로, 이후는 양수로 표기한다. 지역 시간대를 조절하는 작업은 UTC 시각을 기준으로 날짜를 결정한 다음에 처리된다. 예를 들어 milliseconds 인자에 1000을 대입하면 UTC 시각으로 1970년 1월 1일 1초가 되므로 미국 동부 표준시(EST, Eastern Standard Time) 기준으로 따지면 1969년 12월 31일 7:00:01 p.m.이 된다(미 동부 표준시는 UTC보다 다섯 시간 느림).

year

연도를 나타내는 정수. year,...ms 형식을 사용할 때 필요한 인자이다. year에 한 자리 또는 두 자리 숫자를 입력하면 그 값에 1900을 더한 연도로 인식된다(예를 들어 year에 11을 대입하면 2011년이 아니라 1911년이 된다). 2000년 이후의 연도를 입력할 때는 네 자리 숫자를 사용해야 한다(예를 들어 10이 아니라 2010이라고 적어야 한다). 세 자리 연도를 입력하면 A.D. 1000년 이전의 연도로 인식된다. 하지만 year에 음수를 사용하거나 800 미만의 숫자를 사용하면 계산이 제대로 안 될 수도 있다. A.D. 1000년 이전의 날짜를 지정할 때는 milliseconds를 사용하는 생성자를 이용하는 것이 가장 안전하다.

month

월을 지정하기 위한 정수. 1부터 12까지가 아니라 0(1월)부터 11(12월)까지의 숫자로 표기한다. year,...ms 형식의 생성자를 사용할 때 필수적인 인자이다. 0 이상 11 이하의 범위를 벗어나는 숫자를 사용하면 자동으로 그 이전 또는 그 이후의 연도로 넘어간다. 예를 들어 month에 13을 대입하면 다음 연도의 2월로 간주된다.

day

1에서 31까지의 정수를 사용할 수 있으며, 필수적인 인자는 아니다. 이 값을 지정하지 않으면 자동으로 1일로 처리된다. 해당 월의 날짜 범위를 벗어나는 숫자를 사용하면 자동으로 그 이전 혹은 그 이후 달의 날짜로 처리된다. 예를 들어 9월 31일을 입력하면 10월 1일로, 9월 0일을 대입하면 8월 31일로 처리된다.

hours

0(0시)부터 23(오후 11시)까지의 정수로 시를 표현하는 데 쓰이며 적어주지 않아도 무방하다. 이 값을 지정하지 않으면 자동으로 0으로 처

리된다. AM이나 PM은 사용할 수 없다. 범위를 벗어나는 숫자를 입력하면 그 전날 또는 다음날로 처리된다. 예를 들어 hour에 25를 입력하면 자동으로 다음날 오전 1시로 처리된다.

minutes 0부터 59까지의 정수로 분을 표현하는 데 쓰이고 필수적인 인자는 아니다. 이 값을 지정하지 않으면 자동으로 0으로 처리된다. 범위를 벗어나는 숫자를 사용하면 그 전 또는 다음 시간으로 처리된다. 예를 들어 minute에 60을 대입하면 그 다음 시의 1분이 된다.

seconds 0부터 59까지의 정수로 초를 표현하는 데 쓰이며 필수적인 인자는 아니다. 이 값을 지정하지 않으면 자동으로 0이 대입된다. 범위를 벗어나는 숫자를 입력하면 그 전 또는 다음 분의 초로 인식된다. 예를 들어 seconds에 -1을 대입하면 그 전 분의 59초가 된다.

ms 0부터 999까지의 정수로 밀리초를 지정하는 데 쓰이는 인자이며, 필수 인자는 아니다. 이 값을 지정하지 않으면 0이 기본값으로 쓰인다. 범위를 벗어나는 숫자를 입력하면 그 전 또는 그 다음 초로 인식된다. 예를 들어 ms에 1005를 전달하면 다음 초의 5밀리초로 간주된다.

클래스 메소드

UTC() 주어진 UTC 날짜와 1970년 1월 1일 0시와의 시간 차이를 밀리초 단위로 구하는 메소드

메소드

getDate() 날짜(1 이상 31 이하)를 구하는 메소드

getDay() 요일(0(일요일) 이상 6(토요일) 이하의 정수)을 구하는 메소드

getFullYear() 연도를 나타내는 네 자리 정수를 리턴하는 메소드

getHours() 0부터 23까지의 시를 리턴하는 메소드

getMilliseconds() 밀리초를 리턴하는 메소드

getMinutes()

분을 리턴하는 메소드

getMonth()

달(0(1월) 이상 11(12월) 이하의 정수)을 리턴하는 메소드

getSeconds()

초를 리턴하는 메소드

getTime()

내부 형식(1970년 1월 1일 0시 이후로 경과한 시간을 밀리초 단위로 표현한 정수)으로 날짜를 리턴하는 메소드

getTimezoneOffset()

UTC와 지역 시간대 사이의 간격을 분 단위로 리턴하는 메소드

getUTCDate()

UTC 시각을 기준으로 일(날짜)을 리턴하는 메소드

getUTCDay()

UTC 시각을 기준으로 요일을 리턴하는 메소드

getUTCFullYear()

UTC 시각을 기준으로 네 자리 연도를 리턴하는 메소드

getUTCHours()

UTC 시각을 기준으로 시를 리턴하는 메소드

getUTCMilliseconds()

UTC 시각을 기준으로 밀리초를 리턴하는 메소드

getUTCMonth()

UTC 시각을 기준으로 월을 리턴하는 메소드

getUTCSeconds()

UTC 시각을 기준으로 초를 리턴하는 메소드

getYear()

1900년을 기준으로 하는 연도를 리턴하는 메소드

setDate()

일(날짜)을 설정하는 메소드

setFullYear()

네 자리 정수 형식으로 연도를 설정하는 메소드

setHours()

시를 설정하는 메소드

setMilliseconds()

밀리초를 설정하는 메소드

setMinutes()

분을 설정하는 메소드

setMonth()

월을 설정하는 메소드

setSeconds()

초를 설정하는 메소드

setTime() 내부 형식(1970년 1월 1일 0시 이후로 경과한 시간을 밀리초 단위로 표현한 정수)으로 날짜를 설정하는 메소드

setUTCDate()

UTC 시각으로 일(날짜)을 설정하는 메소드

setUTCFullYear()

UTC 시각으로 네 자리 정수 연도를 설정하는 메소드

setUTCHours()

UTC 시각으로 시를 설정하는 메소드

setUTCMilliseconds()

UTC 시각으로 밀리초를 설정하는 메소드

setUTCMinutes()

UTC 시각으로 분을 설정하는 메소드

setUTCMonth()

UTC 시각으로 월을 설정하는 메소드

setUTCSeconds()

UTC 시각으로 초를 설정하는 메소드

setYear() 네 자리 숫자 또는 두 자리 숫자(1900년대만)로 연도를 설정하는 메소드

toString() 날짜를 표현하는 문자열을 리턴하는 메소드

valueOf() 1970년 1월 1일 0시(UTC)와 해당 날짜 사이의 시간을 밀리초 단위로 표현한 정수

설명

Date 클래스의 객체는 시각과 날짜를 알아내거나 임의의 날짜와 시각을 구조화된 형식으로 저장하기 위한 용도로 쓰인다.

액션스크립트에서는 어떤 날짜와 시각을 1970년 1월 1일 0시를 기준으로 하여 밀리초 단위로 표현한다. 만약 그 숫자(밀리초)가 양수이면 1970년 1월 1일 0시 이후를 나타내며, 음수이면 1970년 1월 1일 0시 이전을 나타낸다. 예를 들어 밀리초 값이 10000이면 그 날짜와 시각은 1970년 1월 1일 00:00:10가 되고 -10000이면 1969년 12월 31일 11:59:50가 된다.

하지만 보통 특정 날짜와 시각을 표현할 때 항상 밀리초 값을 계산해야 하는 것은 아니니 너무 걱정하지 않아도 된다. 이런 작업은 액션스크립트 인터프리터에서 모두 처리해 준다. Date 객체를 만들 때는 날짜와 시각을 연도, 월, 일, 시, 분, 초, 밀리초 단위로 입력하기만 하면 된다. 그러면 인터프리터에서 알아서 그 날짜와 시각을 1970년부터의 밀리초 단위로 표현하는 형식으로 변환한다. 또한 현재 시각을 이용하여 Date 객체를 만들 수도 있다. Date 객체를 만들고 나면 Date 클래스의 메소드를 이용하여 연도, 월, 일, 시, 분, 초, 밀리초 값을 알아낼 수도 있고 다른 값으로 설정할 수도 있다.

새로운 Date 객체는 다음 중 한 가지 방법으로 만들 수 있다.

- 아무런 인자도 없이 Date() 생성자를 호출할 수 있다. 이렇게 하면 현재 시각을 나타내는 Date 객체가 생성된다.

- Date() 생성자를 호출할 때 숫자 하나만을 인자로 전달할 수도 있다. 이 숫자는 1970년 1월 1일 0시와 만들고자 하는 날짜 사이의 시간을 밀리초 단위로 나타낸 값이다.
- Date() 생성자를 호출할 때 두 개에서 일곱 개까지의 인자를 전달할 수도 있다. 이 때 연도와 월(여기까지는 필수), 일, 시, 분, 초, 밀리초(여기까지는 옵션)를 인자로 전달한다.

날짜는 객체 내부에 숫자 하나만으로 저장되므로 Date 클래스의 객체에는 속성이 없다. 대신 사람들이 사용하기 좋도록 날짜와 시각의 다양한 성분들을 따로 액세스할 수 있는 메소드가 제공된다. 예를 들어 특정 Date 객체의 월을 알고 싶다면 다음과 같은 명령을 사용하면 된다.

```
myMonth = myDate.getMonth();
```

다음과 같은 표현은 사용할 수 없다.

```
myMonth = myDate.month; // month라는 속성은 없다.
// 대신 메소드를 사용해야 한다.
```

Date 객체의 메소드 중에는 UTC라는 단어가 들어있는 것들이 많이 있는데, 여기서 UTC란 Universal Time Coordinated의 약자로 세계 협정시를 의미한다. 대개 UTC 시각은 그리니치 표준시(GMT: Greenwich Mean Time, 영국 그리니치 천문대를 기준으로 하는 시각)와 똑같다. UTC는 사실 지금까지 역사적으로 의미가 조금씩 바뀌었던 GMT를 대신하는 새로운 표준에 불과하다. UTC 메소드를 이용하면 시간 대별로 시각을 바꿀 필요 없이 직접 UTC를 이용하여 날짜 및 시각에 관련된 작업을 처리할 수 있다. UTC를 사용하지 않는 메소드에서는 지역 시간대를 기준으로 삼는다.

주의 사항

모든 날짜와 시각은 플래시 웹 서버가 아닌 플래시 무비가 실행되는 컴퓨터 운영체제의 설정에 의해 결정되며, 지역에 따라 서로 다른 오프셋을 사용한다. 따라서 사용자 시스템의 날짜 및 시각이 부정확하다면, 액션스크립트에서 쓰이는 날짜와 시각도 정확할 수 없다.

Date() 생성자를 전역 함수로 사용하면 어떤 Date 객체의 toString() 메소드를 실행시켰을 때와 똑같은 형식으로 현재 시각을 나타내는 문자열을 리턴한다는 점에 주의하자.

플래시 5 액션스크립트에서는 자바스크립트와는 달리 어떤 문자열을 Date 객체로 변환할 수 없다.

예제

Date 객체의 합이나 차를 구하면 누적된 시간 또는 시간차를 구할 수 있다. 그레이엄이라는 친구가 1년 미만의 기간을 계획하고 여행을 떠난다고 생각해 보자. 이 친구가 떠난 지 얼마나 되었는지, 그리고 몇일 후에 돌아올지 알아내고 싶다면, 다음과 같은 코드를 사용하면 된다.

```
// now에 현재 시각을 저장한다.
var now = new Date();

// 그레이엄은 2000년 9월 7일에 출발한다(월은 0부터 세기 때문에
// 9월은 9가 아닌 8로 써야 한다).
var departs = new Date(2000,8,7);

// 그레이엄이 돌아올 날짜는 2001년 8월 15일이다.
var returns = new Date(2001,7,15);

// 두 시각을 비교하기 좋도록 밀리초로 변환하고 나서
// 두 날짜 사이의 차가 몇 밀리초인지 알아본다.
var gone = now.getTime() - departs.getTime();

// 두 날짜의 차를 하루에 해당하는 밀리초 값으로 나눈다.
// 이 값으로부터 떠난 지 몇 일 되었는지 확인할 수 있다.
var numDaysGone = Math.floor(gone/86400000);

// 똑같은 방법으로 앞으로 몇 일 있어야 그 친구가 돌아올지 알아본다.
var left = returns.getTime() - now.getTime();
var numDaysLeft = Math.floor(left/86400000);

// 결과를 텍스트 필드에 표시한다.
goneOutput = numDaysGone;
leftOutput = numDaysLeft;
```

어떤 날짜에 다른 날짜를 더하거나 뺄 때는 하루에 해당하는 밀리초 값을 미리 변수에 저장해두는 것이 좋다. 다음과 같은 코드를 이용하면 현재 날짜에 하루를 더할 수 있다.

```
oneDay = 86400000;
now = new Date();
tomorrow = new Date(now.getTime() + oneDay);

// 또한 다음과 같은 식으로 하루를 더할 수도 있다.
now.setTime(now.getTime() + oneDay);
```

날짜를 표현하는 형식을 별도로 만들고 싶다면, 다음과 같이 getDate(), getDay(), getHours()와 같은 메소드의 리턴 값을 이용하여 문자열을 만들어야 한다.

```
// Date 객체를 인자로 전달하면 다음과 같은 형식으로
// 이루어진 문자열을 리턴한다.
// Saturday December 16
function formatDate(theDate) {
    var months = ["January", "February", "March", "April",
                  "May", "June", "July", "August", "September",
                  "October", "November", "December"];
    var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
               "Friday", "Saturday"];
    var dateString = days[theDate.getDay()] + " "
                     + months[theDate.getMonth()] + " "
                     + theDate.getDate();

    return dateString;
}

now = new Date();
trace("Today is " + formatDate(now));
```

getHours() 메소드의 24시간 표기법을 사용하는 리턴 값을 AM과 PM을 사용하는 12시간 표기법으로 변환하고 싶다면, 다음 예와 같은 식으로 하면 된다.

```
// Date 객체를 인자로 전달하면 다음과 같은 형식으로 이루어진 문자열을 리턴
// 한다.
// "2:04PM"
function formatTime(theDate) {
    var hour = theDate.getHours();
    var minute = theDate.getMinutes() > 9 ?
```

```

        theDate.getMinutes() : "0"+ theDate.getMinutes();
    if (hour > 12) {
        var timeString = (hour - 12) + ":" + minute + "PM";
    } else {
        var timeString = hour + ":" + minute + "AM";
    }
    return timeString;
}

now = new Date();
trace("The time is " + formatTime(now));

```

‘13장. 무비 클립’의 ‘클립을 이용하여 시계 만들기’ 부분을 보면 Date 클래스를 이용하여 아날로그 시계를 만드는 예제가 나와 있다. 무비에서 10초 동안 잠시 프로그램 실행을 중단하는 것과 같이 시간을 기준으로 어떤 작업을 처리하는 프로그램을 만들 때는 getTimer()라는 전역 함수를 사용하는 것이 좋다.

참조

Date(), Date.UTC(), getTimer()

Date.getDate() 메소드

날짜를 리턴하는 메소드

버전 플래시 5

문법 `date.getDate()`

리턴 값

date 객체에서 날짜를 나타내는 1 이상 31 이하의 정수

Date.getDay() 메소드

요일을 리턴하는 메소드

버전 플래시 5

문법 `date.getDay()`

리턴 값

date 객체의 요일을 나타내는 0(일요일) 이상 6(토요일) 이하의 정수

예제

다음과 같은 코드를 이용하면 welcomeHeader라는 무비 클립에 요일에 따라 서로 다른 .swf 파일을 가져올 수 있다(일곱 개의 .swf 파일이 sun.swf, mon.swf와 같은 이름으로 저장되어 있어야 한다).

```
now = new Date();
today = now.getDay();
days = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"];
welcomeHeader.loadMovie(days[today] + ".swf");
```

Date.getFullYear() 메소드

네 자리 정수 표현된 연도를 리턴하는 메소드

버전 플래시 5

문법 `date.getFullYear()`

리턴 값

1999나 2000과 같이 date 객체의 연도를 나타내는 네 자리 정수

Date.getHours() 메소드

시를 리턴하는 메소드

버전 플래시 5

문법 `date.getHours()`

리턴 값

date 객체의 시를 나타내는 0(자정) 이상 23(오후 11시) 이하의 정수. A.M.이나 P.M.을 이용하는 표기법은 지원되지 않지만 Date 클래스의 예제에 나온 방법을 이용하여 수동으로 이러한 기능을 구현할 수도 있다.

Date.getMilliseconds() 메소드

밀리초를 리턴하는 메소드

버전 플래시 5

문법 `date.getMilliseconds()`

리턴 값

date 객체의 밀리초를 나타내는 0 이상 999 이하의 정수. 이 메소드에서 리턴되는 값은 1970년과 date 객체가 가리키는 시각 사이의 시간을 밀리초로 표현한 값 (getTime() 참조)이 아니라 주어진 Date 객체에 저장된 시각을 시, 분, 초, 밀리초 형식으로 표현할 때의 밀리초 값이다.

참조

Date.getTime()

Date.getMinutes() 메소드

분을 리턴하는 메소드

버전 플래시 5

문법 `date.getMinutes()`

리턴 값

date 객체의 분을 나타내는 0 이상 59 이하의 정수

Date.getMonth() 메소드

월을 리턴하는 메소드

버전 플래시 5

문법 `date.getMonth()`

리턴 값

date의 월을 나타내는 0(1월) 이상 11(12월) 이하의 정수(1부터 12까지로 표기하지 않는다)

주의 사항

0이 1월이라는 점에 주의하자. getMonth()의 리턴 값은 1이 아니라 0부터 시작된다.

예제

다음과 같은 식으로 `getMonth()`에서 리턴된 값을 영어에서 달을 표기하는 방법으로 변환할 수 있다.

```
var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",  
              "Jul", "Aug", "Sept", "Oct", "Nov", "Dec"];  
myDateObj = new Date();  
trace ("The month is " + months[myDateObj.getMonth()]);
```

Date.getSeconds() 메소드

초를 리턴하는 메소드

버전 플래시 5

문법 `date.getSeconds()`

리턴 값

date 객체의 초를 나타내는 0 이상 59 이하의 정수

Date.getTime() 메소드

1970년 1월 1일과 Date 객체 사이의 시간 차이를
밀리초 단위로 구하는 메소드

버전 플래시 5

문법 `date.getTime()`

리턴 값

date 객체의 시각과 1970년 1월 1일 0시와의 시간 차이를 밀리초 단위로 계산한 정수. 1970년 1월 1일 이후이면 양수, 이전이면 음수가 리턴된다.

설명

내부적으로 모든 Date 객체는 그 객체에 저장된 시각과 1970년 1월 1일 0시와의 시간 차이를 밀리초 단위로 표현한 정수만으로 표현된다. `getTime()` 메소드를 이용하면 그 값을 알아낼 수 있으며 이 리턴 값을 이용하여 두 Date 객체 사이의 시간 차이를 알아내거나 새로운 Date 객체를 만들 수 있다.

예제

무비의 10번째 프레임에 다음과 같은 코드를 입력했다고 가정하자.

```
time1 = new Date();
```

그리고 20번째 프레임에 다음과 같은 코드를 입력했다고 가정하자.

```
time2 = new Date();
```

그리고 나서 다음과 같은 코드를 실행시키면 무비의 10번째 프레임과 20번째 프레임 사이의 시간 차이를 알 수 있다.

```
elapsedTime = time2.getTime() - time1.getTime();
```

전역 함수인 `getTime()` 함수를 이용하여 무비에서 경과한 시간을 알아낼 수도 있다.

참조

`Date.setTime()`, `getTime()`

Date.getTimezoneOffset() 메소드

지역 시간대와 UTC(GMT) 사이의 시간 차이를
분 단위로 표현한 값을 리턴하는 메소드

버전 플래시 5

문법 `date.getTimezoneOffset()`

리턴 값

UTC(GMT) 시각과 지역 시간대의 시간 차이를 분 단위로 표현한 값. 지역 시간대가 UTC보다 느리면 양수, 빠르면 음수가 리턴된다. 일광절약제를 이용하는 경우에는 일광절약제 시행에 따른 시각 변화도 포함된다.

예제

미국 동부 표준시(EDT)를 기준으로 일광절약제가 실시되는 기간에 다음과 같은 코드를 실행시키면 240(네 시간)이 리턴된다(한국은 UTC보다 아홉 시간 빠르므로 -540이 리턴된다).

```
myDate = new Date();  
trace(myDate.getTimezoneOffset()); // 240이 리턴된다.
```

하지만 일광절약제가 실시되지 않는 기간에 위 코드를 실행시키면 300(다섯 시간)이 리턴된다.

Date.getUTCDate() 메소드

UTC를 기준으로 날짜를 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCDate()`

리턴 값

UTC를 기준으로 date 객체에 저장된 날짜를 나타내는 정수 값

Date.getUTCDay() 메소드

요일(UTC 기준)을 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCDay()`

리턴 값

UTC를 기준으로 date 객체에 저장된 날짜의 요일을 나타내는 0(일요일) 이상 6(토요일) 이하의 정수

Date.getUTCFullYear() 메소드

네 자리 연도(UTC 기준)를 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCFullYear()`

리턴 값

1999나 2000과 같이 UTC를 기준으로 date에 저장된 연도를 나타내는 네 자리 정수.

Date.getUTCHours() 메소드

시(UTC 기준)를 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCHours()`**리턴 값**

UTC를 기준으로 date 객체에 저장된 날짜의 시를 나타내는 0(자정) 이상 23(오후 11시) 이하의 정수

Date.getUTCMilliseconds() 메소드

밀리초(UTC 기준)를 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCMilliseconds()`**리턴 값**

UTC를 기준으로 date 객체에 저장된 날짜의 밀리초를 나타내는 0 이상 999 이하의 정수

Date.getUTCMinutes()

분(UTC 기준)을 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCMinutes()`**리턴 값**

UTC를 기준으로 date 객체에 저장된 날짜의 분을 나타내는 0 이상 59 이하의 정수

Date.getUTCMonth()

월(UTC 기준)을 리턴하는 메소드

버전 플래시 5

문법 `date.getUTCMonth()`

리턴 값

UTC를 기준으로 date 객체에 저장된 날짜의 월을 나타내는 0(1월) 이상 11(12월) 이하의 정수(1부터 12까지가 아님)

주의 사항

0이 1월이라는 점에 주의하자. `getMonth()`의 리턴 값은 1이 아니라 0부터 시작된다.

Date.getUTCSeconds()

초(UTC 기준)를 리턴하는 메소드

버전 플래시 5**문법** `date.getUTCSeconds()`**리턴 값**

UTC를 기준으로 date 객체에 저장된 날짜의 초를 나타내는 0 이상 59 이하의 정수

Date.getYear() 메소드

1900년을 기준으로 하는 연도

버전 플래시 5**문법** `date.getYear()`**리턴 값**

`date.getFullYear()` -1900과 같은 값. 예를 들어 1999년의 어떤 날짜에 대해 `getYear()` 메소드를 실행시키면 99가, 2001년의 어떤 날짜에 대해 실행시키면 101이, 1800년의 어떤 날짜에 대해 실행시키면 -100이 리턴된다. 주로 1900년대에 해당하는 연도를 다룰 때 유용하다.

Date.setDate() 메소드

날짜를 설정하는 메소드

버전 플래시 5**문법** `date.setDate(day)`**인자**

day date 객체에 들어갈 새로운 날짜를 나타내는 1 이상 31 이하의 정수. 주어진 달에 있는 날수보다 큰 값을 입력하면 다음 달로 넘어간다. 예를 들어 현재 month가 8(즉 9월)인데 day에 31을 입력하면 10월 1일이 된다. 따라서 month가 9(10월)가 되고 day는 1이 된다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조`Date.getDate()`

Date.setFullYear() 메소드

네 자리 정수 형태로 연도를 설정하는 메소드

버전 플래시 5**문법** `date.setFullYear(year, month, day)`**인자**

year 1999나 2000과 같이 date 객체에 저장할 날짜의 연도를 나타내는 네 자리 정수

month date 객체에 저장할 날짜의 월을 나타내는 0(1월) 이상 11(12월) 이하의 정수(1부터 12까지가 아님). 이 값을 지정하지 않으면 0이 기본값으로 쓰인다(옵션).

day date 객체에 저장할 날짜를 나타내는 1 이상 31 이하의 정수. 이 값을 지정하지 않으면 1이 기본값으로 쓰인다(옵션).

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조

`Date.setYear()`, `Date.getFullYear()`

Date.setHours() 메소드

시를 설정하는 메소드

버전 플래시 5

문법 `date.setHours(hour)`

인자

hour `date` 객체에 저장할 날짜의 시에 해당하는 0(자정) 이상 23(오후 11시) 이하의 정수

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조

`Date.getHours()`

Date.setMilliseconds() 메소드

밀리초를 설정하는 메소드

버전 플래시 5

문법 `date.setMilliseconds(ms)`

인자

ms `date` 객체에 저장할 시각에 해당하는 밀리초를 나타내는 0 이상 999 이하의 정수(1970년과의 시간 차이를 밀리초 단위로 나타낸 값이 아님). 999보다 크거나 0보다 작은 정수를 사용하면 `date` 객체의 초 단위도 변경된다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조

`Date.getMilliseconds()`, `Date.setTime()`

Date.setMinutes() 메소드

분을 설정하는 메소드

버전 플래시 5

문법 `date.setMinutes(minute)`

인자

minute `date` 객체에 저장할 시각의 분을 나타내는 0 이상 59 이하의 정수. 59보다 크거나 0보다 작은 정수를 사용하면 `date` 객체의 시도 바뀐다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조

`Date.getMinutes()`

Date.setMonth() 메소드

월을 설정하는 메소드

버전 플래시 5

문법 `date.setMonth(month)`

인자

month `date` 객체에 저장할 날짜의 월을 나타내는 0(1월) 이상 11(12월) 이하의 정수(1부터 12까지가 아님). 11보다 크거나 0보다 작은 숫자를 사용하면 연도도 함께 바뀐다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

주의 사항

1월이 1로 표시된다고 착각하지 않도록 주의하자. 월은 1이 아니라 0부터 시작한다.

참조

`Date.getMonth()`

Date.setSeconds() 메소드

초를 설정하는 메소드

버전 플래시 5

문법 `date.setSeconds (second)`

인자

second `date` 객체에 저장할 날짜의 초를 나타내는 0 이상 59 이하의 정수. 59보다 크거나 0보다 작은 값을 전달하면 분도 함께 바뀐다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

참조

`Date.getSeconds()`

Date.setTime() 메소드

1970년 1월 1일 0시와의 시간 차이를 밀리초 단위로 나타낸 정수를 기반으로 새로운 날짜를 설정한다.

버전 플래시 5

문법 `Date.setTime(milliseconds)`

인자

milliseconds

새로운 날짜와 1970년 1월 1일 0시와의 시간 차이를 밀리초 단위로 나타낸 값. 1970년 1월 1일 이후는 양수, 이전은 음수로 표시한다.

리턴 값

milliseconds 값

설명

Date 객체 내부에서는 모든 날짜와 시각을 1970년 1월 1일과의 시간 차이를 밀리초 단위로 나타낸 값으로 표기한다. setTime() 메소드에서는 내부적으로 사용하는 밀리초 단위 표기법으로 새로운 날짜를 지정할 수 있다. 1970년과의 시간 차이를 밀리초 단위로 나타낸 값을 이용하여 날짜를 설정하는 방법은 getTime() 메소드를 이용하여 여러 날짜와 시각 사이의 시간 차이를 알아낼 때 유용하다.

예제

setTime()을 getTime()과 같이 이용하면 시간 차이에 해당하는 밀리초 값을 더하거나 빼서 원래 있던 날짜를 변경할 수 있다. 예를 들어 다음과 코드를 이용하면 한 시간을 더할 수 있다.

```
now = new Date();
now.setTime(now.getTime() + 3600000);
```

또한 다음 코드를 이용하여 하루를 더할 수도 있다.

```
now = new Date();
now.setTime(now.getTime() + 86400000);
```

한 시간, 하루를 밀리초 단위로 나타낸 값을 변수에 따로 저장해 두면 프로그램의 가독성을 향상시킬 수 있다.

```
oneDay = 86400000;
oneHour = 3600000;
now = new Date();
// 하루하고 세 시간을 뺀다.
now.setTime(now.getTime() - oneDay - (3 * oneHour));
```

참조

`Date.getTime()`, `Date.setMilliseconds()`, `Date.UTC()`, `getTimer()`

Date.setUTCDate() 메소드

날짜(UTC 기준)를 설정하는 메소드

버전 플래시 5

문법 `date.setUTCDate(day)`

인자

day UTC를 기준으로 date 객체의 일(날짜)에 대입할 1 이상 31 이하의 정수. 그 달의 날짜에 맞지 않는 값을 사용하면 월도 같이 바뀐다. 예를 들어 month 값이 8(9월)일 때 day 값에 31을 전달하면 10월 1일로 간주된다. 따라서 month 값은 9(10월)가 되고 day 값은 1이 된다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setUTCFullYear() 메소드

네 자리수 정수로 연도(UTC 기준)를 설정하는 메소드

버전 플래시 5

문법 `date.setUTCFullYear(year)`

인자

year 1999나 2000과 같이 UTC를 기준으로 date 객체의 연도로 대입할 네 자리 정수

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setUTCHour() 메소드

시(UTC 기준)를 설정하는 메소드

버전 플래시 5**문법** `date.setUTCHour(hour)`**인자****hour** UTC를 기준으로 date 객체의 시에 대입할 0(자정) 이상 23(오후 11시) 이하의 정수**리턴 값**

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setUTCMilliseconds() 메소드

밀리초(UTC 기준)를 설정하는 메소드

버전 플래시 5**문법** `date.setUTCMilliseconds(ms)`**인자****ms** UTC를 기준으로 date 객체의 밀리초에 대입할 0 이상 999 이하의 정수**리턴 값**

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setUTCMinutes() 메소드

분(UTC 기준)을 설정하는 메소드

버전 플래시 5**문법** `date.setUTCMinutes(minute)`

인자

minute UTC를 기준으로 date 객체의 분에 대입할 0 이상 59 이하의 정수

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setUTCMonth() 메소드

월(UTC 기준)을 설정하는 메소드

버전 플래시 5

문법 `date.setUTCMonth(month)`

인자

month UTC를 기준으로 date 객체의 월에 대입할 0(1월) 이상 11(12월) 이하의 정수(1 이상 12 이하가 아님)

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

주의 사항

1월은 1로 표기하지 않는다. 월은 1이 아니라 0부터 시작한다는 점에 주의하자.

Date.setUTCSeconds() 메소드

초(UTC 기준)를 설정하는 메소드

버전 플래시 5

문법 `date.setUTCSeconds(second)`

인자

second UTC를 기준으로 date 객체의 초에 대입할 0 이상 59 이하의 정수

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

Date.setYear() 메소드

1900년을 기준으로 연도를 설정하는 메소드

버전 플래시 5

문법 `date.setYear(year, month, day)`

인자

year date 객체의 연도를 나타내는 정수로 반드시 지정해주어야 한다. 한 자리 또는 두 자리 정수를 사용하면 1900년대의 연도로 간주된다. 예를 들어 1은 1901년, 99는 1999년을 나타낸다. 세 자리 연도를 입력하면 A.D. 1000년 이전의 연도로 간주된다. 2000년 이후의 연도를 표기할 때는 네 자리 숫자를 사용해야 한다.

month date 객체의 월을 나타내는 0(1월) 이상 11(12월) 이하의 정수(1부터 12까지가 아님)(옵션)

day date 객체의 일을 나타내는 1 이상 31 이하의 정수(옵션)

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

설명

setYear()는 한 자리 및 두 자리 정수는 1900년대의 연도로 간주한다는 점을 제외하면 setFullYear() 메소드와 똑같다(setFullYear() 메소드에서는 year 값을 서기 연도로 간주한다).

참조

Date.getYear(), Date.setFullYear()

Date.toString() 메소드

날짜와 시각을 나타내는 문자열을 리턴하는 메소드

버전 플래시 5**문법** `date.toString()`**리턴 값**

`date` 객체에 저장된 날짜와 시각을 일반적으로 사용하는 형태(UTC 오프셋도 출력)로 나타낸 문자열. 날짜와 시각을 원하는 형식으로 표현하고 싶다면 일, 시, 분, 초와 같은 값을 구하는 메소드를 별도로 사용하여 문자열을 만들어야 한다(Date 클래스 참조).

예제

```
trace (myDate.toString()); // 다음과 같은 형식으로 날짜를 출력한다.
// Wed Sep 15 12:11:33 GMT-0400 1999
```

Date.UTC() 클래스 메소드

주어진 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 값을 리턴하는 메소드

버전 플래시 5**문법** `Date.UTC(year, month, day, hours, minutes, seconds, ms)`**인자**

year,...ms 지역 시각이 아닌 UTC 시각으로 연도, 월, 일, 시, 분, 초, 밀리초를 지정하는 인자. 각 인자에 대한 설명은 `Date()` 생성자 부분에 나와 있다.

리턴 값

새로운 날짜와 1970년 1월 1일 0시 사이의 시간 차이를 밀리초 단위로 나타낸 정수

설명

`Date.UTC()` 메소드에서는 `Date()` 생성자와 똑같은 인자를 사용하지만 이 메소드에서는 주어진 날짜와 시각을 나타내는 객체를 리턴하지 않고 1970년과의 시간 차이를 밀리초 단위로 나타낸 값을 리턴한다. 이렇게 리턴된 값은 보통 UTC를 기준

으로 새로운 Date 객체를 만들거나 setTime() 메소드를 통해 어떤 객체에 UTC 시각을 대입하는 데 쓰인다.

예제

다음과 같은 코드를 이용하면 1970년 1월 1일 자정과 2000년 1월 1일 자정 사이의 시간 차이를 밀리초 단위로 알아낼 수 있다.

```
trace(Date.UTC(2000, 0) + " milliseconds passed between 1970 and 2000.");
// "946684800000 milliseconds passed between 1970 and 2000."가 출력된다.
```

또한 다음과 같은 방법으로 UTC를 기준으로 하는 Date 객체를 만들 수도 있다.

```
nowUTC = new Date(Date.UTC(2000, 0));
```

위와 같은 코드를 EST(미국 동부 표준시, UTC보다 다섯 시간 느림)를 기준으로 호출하면 nowUTC에는 1999년 12월 31일 오후 7시라는 지역 시간이 기록된다. UTC를 기준으로 하지 않는 getHours() 메소드를 이용하면 위에서 설정한 nowUTC 객체의 지역 시간대를 기준으로 한 시인 19(오후 7시)가 리턴된다.

```
trace(nowUTC.getHours()); // 19가 출력된다.
```

하지만 UTC 메소드인 getUTCHours()를 이용하면 0(자정)이 리턴된다.

```
trace(nowUTC.getUTCHours()); // 0이 출력된다.
```

참조

Date(), Date.setTime(), Date 클래스

Date.valueOf() 메소드

Date 객체에 저장된 시각과 1970년 1월 1일 0시 사이의 시간 간격을 밀리초 단위로 나타낸 값을 리턴하는 메소드

버전 플래시 5

문법 date.valueOf()

리턴 값

date 객체와 1970년 1월 1일 사이의 시간 간격을 밀리초 단위로 나타낸 정수. 1970년 1월 1일 이후이면 양수, 이전이면 음수가 리턴된다.

설명

Date.valueOf()와 Date.getTime() 메소드 중 어떤 것을 사용하더라도 결과는 똑같다.

참조

Date.toString()

duplicateMovieClip() 전역 함수

무비 클립 복사

버전 플래시 4 이후

문법 duplicateMovieClip(*target*, *newName*, *depth*)

인자

target 복사할 무비 클립(씨앗 클립이라고도 함)을 가리키는 경로를 나타내는 문자열. 중첩된 클립은 duplicateMovieClip("_root.myClip", "myClip2", 0)과 같은 식으로 점 표기법을 이용하여 표현할 수 있다. 무비 클립 레퍼런스를 문자열이 들어갈 자리에 사용하면 자동으로 경로로 변환되므로, target 자리에는 duplicateMovieClip(myClip, "myClip2", 0)과 같이 무비 클립 레퍼런스를 사용해도 무방하다.

newName 복사본의 이름으로 사용할 문자열. newName에 사용할 문자열에는 '14장. 렉시컬 구조'의 '인식자' 부분에 나온 규칙이 적용된다.

depth 클립 스택에서 복사된 클립을 추가할 레벨을 나타내는 정수. 낮은 레벨에 있는 클립은 높은 레벨에 있는 클립보다 뒤에 있는 것처럼 보인다. 스택에 있는 무비 클립 중에 depth 값이 가장 큰 클립은 그보다 작은 depth 값을 가진 클립보다 위에 놓인다. 예를 들어 depth가 -1인 클립은 depth가 0인 클립 밑에 출력되고 depth가 1인 클립은 depth가 0인 클립 위에 출력된다. 주어진 depth 값에 이미 다른 클립

이 있다면 원래 있던 클립은 삭제되고 새로운 클립이 그 자리에 들어간다. 자세한 설명은 13장에 나와 있다. depth 값에 음수를 대입해도 프로그램이 실행되긴 하지만 액션스크립트에서 공식적으로 음수 depth 값을 지원하는 것은 아니다. 플래시의 차기 버전에서 depth에 음수 값을 사용하지 못하도록 만들 수 있기 때문에, 나중에도 호환성을 유지하기 위해서는 depth 값에 0 이상의 정수만을 사용하는 것이 좋다.

설명

`duplicateMovieClip()` 함수는 무비를 재생할 때 무비 클립을 새로 만드는 방법 중 하나이다(나머지 한 가지 방법은 `attachMovie()`를 이용하는 방법이다). `duplicateMovieClip()` 함수를 실행시키면 target 클립과 똑같은 복사본이 하나 생기고 그 복사본은 depth 번째 레이어에 target 클립과 같은 클립 스택에 추가된다. 이렇게 만든 클립의 복사본은 복사 당시에 target이 있던 프레임과는 상관없이 무조건 1번 프레임부터 재생된다.

클립 복사본은 target에 적용된 모든 변환(회전, 확대/축소 등)을 그대로 물려받지만 target 클립의 타임라인 변수는 복사되지 않는다. `duplicateMovieClip()`이라는 함수는 무비 클립 메소드로 사용할 수도 있는데, 그 경우에는 target 인자를 사용하지 않는다.

예제

```
// ball 클립을 복사하여 ball2라는 클립을 만든다.
duplicateMovieClip(ball, "ball2", 0);
// 새로운 ball2 클립을 옮겨서 화면에 드러나도록 만든다.
ball2._x += 100;
```

참조

`MovieClip.duplicateMovieClip()`, `removeMovieClip()`, 13장의 ‘`duplicateMovieClip()`을 이용하여 만드는 방법’, ‘`duplicateMovieClip()`을 이용하여 생성된 클립이 스택에 추가되는 과정’, ‘메소드와 전역 함수가 겹치는 문제’

escape() 전역 함수

네트워크를 통해 안전하게 전송할 수 있도록
문자열을 인코딩

버전 플래시 5

문법 `escape(string)`

인자

string 인코딩할 문자열(또는 문자열 표현식)

리턴 값

string을 URL 인코딩한 값

설명

`escape()` 함수에서는 주어진 문자열을 이용하여 새로운 인코딩된 문자열을 만든다. 새로운 문자열에서 숫자나 A-Z, a-z에 해당하는 기본 알파벳을 제외한 모든 글자는 16진수 이스케이프 시퀀스로 변환된다. 이렇게 변환된 16진수 이스케이프 시퀀스는 `%xx` 형태이며 이 때 쓰이는 `xx` 값은 Latin 1 문자 세트의 코드 포인트 값이다. Shift-JIS의 2바이트 문자는 `%xx%xx`와 같이 두 개의 16진수 이스케이프 시퀀스로 변환된다.

`escape()` 함수를 이용하면 문자열을 손쉽게 URL 인코딩할 수 있다(단 스페이스가 `+`가 아닌 `%20`으로 변환된다는 점은 조금 다르다). 플래시 무비에서 서버 애플리케이션에 정보를 전송하거나 브라우저용 쿠키를 보낼 때도 `escape()` 함수를 종종 사용한다.

인코딩된 문자열을 디코딩할 때는 `unescape()`라는 전역 함수를 사용한다.

예제

```
var phoneNumber = "(222) 515-1212"
escape(phoneNumber); // %28222%29%20515%2D1212로 변환된다.
```

참조

`unescape()`, '부록 B. 라틴 1 문자 범주 및 키코드'

eval() 전역 함수

문자열을 인식자로 변환

버전 플래시 4 이후**문법** `eval(stringExpression)`**인자*****stringExpression***

문자열 또는 문자열 표현식. 어떤 인식자의 이름과 같아야 한다.

리턴 값

`stringExpression`으로 표현되는 변수 값 또는 `stringExpression`으로 표현되는 객체 레퍼런스, 무비 클립 또는 함수. `stringExpression`이라는 이름을 가지는 변수나 무비 클립이 없으면 `undefined`를 리턴한다.

설명

`eval()` 함수를 이용하면 문자열 텍스트를 이용하여 인식자에 대한 동적인 레퍼런스를 만들 수 있다. `eval()`에서는 문자열을 변수, 무비 클립, 객체 속성과 같은 인식자로 변환하고 그 인식자의 값을 리턴한다. 예를 들어 아래 코드에서는 `eval()`의 리턴 값을 이용하여 변수 값을 설정한다.

```
name1 = "Kathy";
count = 1;
currentName = eval("name" + count);
// currentName을 "Kathy"로 설정한다.
```

다음과 같이 `star1`이라는 무비 클립 이름을 동적으로 만들어서 제어할 수도 있다.

```
eval("star" + count)._x += 100;
// star1 무비 클립을 오른쪽으로 100픽셀 이동
```

또한 `eval()` 함수를 왼쪽 피연산자로 사용하여 어떤 값을 인식자에 대입할 수도 있다.

```
eval("name" + count) = "Simone";
// name1을 "Simone"으로 설정한다.
```

하지만 자바스크립트의 `eval()` 함수와는 달리 액션스크립트의 `eval()` 함수에서는 임의의 코드 블록을 문자열로 입력받아서 컴파일하고 실행할 수 없다. `eval()` 함수의 기능을 모두 지원하려면 액션스크립트 컴파일러를 플레이어에 포함시켜야 하는데 그러면 플레이어의 크기가 너무 커지기 때문이다.

주의 사항

플래시 5에서는 동적으로 변수를 액세스할 때도 `eval()`을 거의 사용하지 않는다. 여러개의 데이터를 관리할 때는 변수나 객체를 대신 사용한다.

예제

`eval()` 함수는 `MovieClip._droptarget` 문자열 속성을 무비 클립 객체 레퍼런스로 변환할 때 많이 사용한다. 아래 예제에서는 만화 캐릭터의 얼굴을 나타내는 클립이 여러 개 있다고 가정한다. 사용자가 `food` 클립을 화면에 떨어뜨리면 `_droptarget`을 이용하여 그 얼굴에 대한 경로를 알아낸 후 `eval()`을 이용하여 그 얼굴에 대한 객체 레퍼런스를 구한다. 그리고 나서 그 얼굴을 입 속에 음식이 들어있는 얼굴을 보여주는 프레임으로 보낸다.

```
// _droptarget 문자열을 레퍼런스로 변환한다.
theFace = eval(food._droptarget);
// 변환된 레퍼런스를 이용하여 얼굴 클립을 제어한다.
theFace.gotoAndStop("full");
```

참조

4장의 'eval을 이용한 문자열에 있는 코드 실행'

`_focusrect` 전역 속성

키보드로 활성화된 버튼에서 쓰이는 강조 상태 제어

버전	플래시 4 이후
문법	<code>_focusrect</code>
액세스	읽기/쓰기

설명

플래시에서 마우스가 버튼 위를 지나가면 버튼의 Over 상태에 해당하는 내용이 화면에 표시된다. 사용자가 탭 키를 누르면 버튼에 키보드 포커스가 적용될 수도 있다. 어떤 버튼에 키보드 포커스가 적용되어 있으면 플래시에서는 그 버튼에 노란색 직사각형 테두리를 두르는데 이렇게 하면 미관상 좋지 않을 수도 있다. `_focusrect` 전역 속성을 이용하여 이와 같이 노란색 테두리가 생기는 강조 상태를 사용하지 않을 수 있다.

```
_focusrect = false;
```

`_focusrect`를 `false`로 설정하면 키보드 포커스가 적용된 버튼의 Over 상태를 화면에 표시한다. `_focusrect`가 `true`(기본값)이면 노란색 직사각형이 표시된다.

주의 사항

`_focusrect`는 부울형으로 쓰이지만 실제로는 부울이 아닌 숫자가 저장된다. `_focusrect`의 값을 직접 확인해 보면 1(`true`에 해당함) 또는 0(`false`에 해당함)이 리턴된다.

fscommand() 전역 함수

독립형 플레이어 또는 플레이어의
호스트 애플리케이션에 메시지를 보냄

버전 플래시 3 이후(플래시 5에서는 `trapallkeys`를 사용할 수 있음)

문법 `fscommand(command, arguments)`

인자

command 호스트 애플리케이션으로 전달할 문자열. 보통 자바스크립트 함수의 이름을 전달한다.

arguments 호스트 애플리케이션으로 전달할 문자열. 보통 `command`라는 이름의 함수에 전달할 인자를 `arguments`로 전달한다.

설명

`fscommand()` 함수를 이용하면 플래시 무비와 독립형 플레이어 또는 플레이어의 호스트 애플리케이션(웹 브라우저나 매크로미디어 디렉터와 같은 플래시 플레이어

가 실행되는 환경)과 정보를 주고받을 수 있다. `fscommand()`는 보통 다음과 같은 세 가지 방법으로 쓰인다.

- 독립형 플래시 플레이어에 내장된 몇 가지 명령을 전달할 때
- 웹 브라우저에 있는 자바스크립트나 VB스크립트같은 스크립트 언어에 명령을 전달할 때
- 매크로미디어 디렉터 무비의 링고와 정보를 주고받을 때

독립형 플레이어에서 `fscommand()`를 사용하면 [표 R-5]에 나온 것과 같은 내장 명령과 인자를 전달할 수 있다.

[표 R-5] 독립형 플레이어에서 사용할 수 있는 *command*와 *argument* 인자

명령	인자	설명
"allowscale"	"true" 또는 "false"	"false"로 설정하면 플레이어 창 크기를 변경해도 무비 크기가 바뀌지 않는다. 보통 "fullscreen" 옵션과 함께 사용하여 전체 화면 상태에서 무비 크기는 바뀌지 않도록 할 때 사용한다.
"exec"	"application_name"	외부 프로그램을 실행시킨다. application_name 문자열은 애플리케이션의 경로를 나타낸다. application_name을 "C:/WINDOWS/NOTEPAD.EXE"와 같이 절대 경로로 지정하지 않으면 플래시 무비에 대한 상대 경로로 간주한다. 경로를 지정할 때는 역슬래시(\)가 아니라 그냥 슬래시(/)를 사용한다는 점에 주의하자.
"fullscreen"	"true" 또는 "false"	"true"이면 플레이어 창이 전체 화면 상태로 바뀐다.
"quit"	적용할 수 없음	무비를 닫고 플래시 프로젝트(독립형 플레이어)를 종료한다.
"showmenu"	"true" 또는 "false"	"false"로 설정하면 플레이어 메뉴를 모두 없애고 About Macromedia Flash Player 옵션만 남긴다. 메뉴는 오른쪽 클릭(윈도우) 또는 Ctrl-클릭(매킨토시)으로 액세스할 수 있다.
"trapallkeys"	"true" 또는 "false"	"true"로 설정하면 어떤 키를 누르더라도(단축키를 눌러도) 그 내용이 플래시 무비로만 전달된다. trapallkeys를 이용하면 독립형 플레이어에서 모든 제어 키(전체 화면 모드로 바꾸는 Ctrl-F나 Command-F, 프로그램을 종료시키기 위한 Ctrl-Q나 Command-Q, 전체 화면 모드에서 나오기 위한 Esc 등)를 사용할 수 없도록 만들 수 있다(플래시 5에서 추가된 기능).

브라우저에서 무비를 실행할 때 `fscommand()` 함수를 이용하면 그 무비를 포함하는 HTML 페이지에 있는 자바스크립트 함수(넷스케이프) 또는 VB스크립트 함수(인터넷 익스플로러)를 실행할 수 있다. 이러한 함수의 이름은 일반적으로 `movieID_DoFSCommand`와 같은 형식으로 정해지는데, 이 때 `movieID`는 HTML 문서에서 지정한 무비의 OBJECT ID 속성(인터넷 익스플로러)이나 EMBED NAME 속성(넷스케이프)으로 지정한 이름이다. `movieID_DoFSCommand()`를 호출하면 `fscommand()`의 `command`와 `arguments` 매개변수가 `movieID_DoFSCommand()` 함수의 인자로 전달된다. 만약 `movieID_DoFSCommand()` 함수가 호스트 페이지에 없으면 오류 메시지 없이 그냥 넘어간다.

넷스케이프에서 `fscommand()`가 작동하려면 무비의 EMBED 태그의 `swLiveConnect` 속성을 "true"로 설정해야 한다.

```
<EMBED
  NAME="testmovie"
  SRC="myMovie.swf"
  WIDTH="100%"
  HEIGHT="100%"
  swLiveConnect="true"
  PLUGINSOURCE="http://www.macromedia.com/go/flashplayer/">
</EMBED>
```

주의 사항

다음과 같은 시스템에서는 `fscommand()`를 이용하여 브라우저에 정보를 전달할 수 없다.

- 매킨토시 OS의 인터넷 익스플로러
- 매킨토시 68K 시리즈에서 실행되는 브라우저
- 윈도우 3.1에서 실행되는 브라우저
- 넷스케이프 6

플레이어에서 디렉터 무비와 데이터를 주고받을 때 `fscommand()`를 사용하는 것이 항상 최선의 방법은 아니다. 디렉터에서는 `getURL()` 함수를 사용하면서 `event: lingo: 프로토콜`을 사용하는 것이 좋다. 자세한 내용은 `getURL()` 부분이나 아래에 나와 있는 매크로미디어 테크 노트에서 찾아볼 수 있다.

http://www.macromedia.com/support/director/ts/documents/flash_xtra_sending_events.htm

http://www.macromedia.com/support/director/ts/documents/flash_xtra_lingo.htm

http://www.macromedia.com/support/director/ts/documents/flash_tips.htm

예제

독립형 플레이어를 종료시킬 때는 다음과 같은 코드를 사용하면 된다.

```
fscommand("quit");
```

독립형 플레이어를 전체 화면 모드로 실행시키려면 다음과 같이 하면 된다.

```
fscommand("fullscreen", "true");
```

독립형 플레이어에서 전체 화면 모드 상태로 무비 크기는 원래 크기대로 유지한 채 무비를 재생하려면 다음과 같이 하면 된다.

```
fscommand("fullscreen", "true");  
fscommand("allowscale", "false");
```

무비를 웹 브라우저 창에서 전체 화면 모드로 실행시키고 싶다면 <http://www.moock.org/webdesign/flash/launchwindow/fullscreen> 부분을 참조하기 바란다.

다음과 같이 하면 대부분의 윈도우 시스템에 있는 메모장을 실행시킬 수 있다.

```
fscommand("exec", "C:/WINDOWS/NOTEPAD.EXE");
```

아래 코드는 무비의 fscommand()로 실행할 자바스크립트와 VB스크립트가 포함된 HTML 페이지의 예제다. 이 VB스크립트 함수에서는 자바스크립트 함수를 호출하는데, 이렇게 하면 자바스크립트 함수 하나만으로 인터넷 익스플로러와 넷스케이프에서 모두 실행시킬 수 있다.

```
<HTML>  
<HEAD>  
<TITLE>fscommand demo</TITLE>  
  
<SCRIPT LANGUAGE="JavaScript">
```

```
<!--
function testmovie_DoFSCommand(command, args) {
    alert("Here's the Flash message " + command + ", " + args);
}
//-->
</SCRIPT>

<SCRIPT LANGUAGE="VBScript">
<!--
Sub testmovie_FSCommand(ByVal command, ByVal args)
    call testmovie_DoFSCommand(command, args)
end sub
//-->
</SCRIPT>

</HEAD>

<BODY BGCOLOR="#FFFFFF">

<OBJECT
    ID="testmovie"
    CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    WIDTH="100%"
    HEIGHT="100%"

    CODEBASE="http://download.macromedia.com/pub/shockwave/cabs/flash/swfl
ash.cab">
    <PARAM NAME="MOVIE" VALUE="flash-to-javascript.swf">

    <EMBED
        NAME="testmovie"
        SRC="flash-to-javascript.swf"
        WIDTH="100%"
        HEIGHT="100%"
        swLiveConnect="true"
        PLUGINSPACE="http://www.macromedia.com/go/flashplayer/">
    </EMBED>
</OBJECT>

</BODY>
</HTML>
```

flash-to-javascript.swf 무비에서는 다음과 같은 코드를 이용하여 앞에 나온 testmovie_DoFSCommand() 자바스크립트 함수를 실행시킨다.

```
fscommand("hello", "world");
```

참조

fscommand()에 대한 자세한 내용과 자바스크립트를 이용하여 플래시를 제어하는 내용에 대한 정보는 아래 사이트에서 구할 수 있다.

<http://www.moock.org/webdesign/flash/fscommand>

<http://www.macromedia.com/support/flash/publishexport/scriptingwithflash>

getProperty() 전역 함수

무비 클립 속성의 값을 구하는 함수

버전 플래시 4(플래시 5에서는 쓰이지 않음)

문법 `getProperty(movieClip, property)`

인자

movieClip 무비 클립을 가리키는 경로에 해당하는 문자열 표현식. 플래시 5에서는 무비 클립 레퍼런스를 사용해도 된다. 무비 클립 레퍼런스를 문자열 자리에 사용하면 자동으로 경로로 변환되기 때문이다.

property 값을 구할 내장 속성의 이름. 문자열이 아닌 인식자를 사용해야 한다(예: “_x”가 아니라 _x를 사용해야 한다).

리턴 값

movieClip의 property 값

설명

getProperty() 함수를 이용하면 무비 클립의 내장 속성 값을 구할 수 있다. 플래시 4에서는 객체 속성을 액세스하는 방법이 getProperty()밖에 없었지만, 플래시 5 이후에서는 속성을 액세스할 때 점 연산자(.)나 중괄호 연산자({})를 이용하는 것이 좋다.

예제

아래 코드는 모두 메인 무비 타임라인에 있는 ball이라는 무비 클립의 _x 속성을 구하는 코드이다.

```

getProperty(ball, _x);           // 무비 클립의 상대 레퍼런스를 이용
getProperty(_root.ball, _x);     // 무비 클립의 절대 레퍼런스를 이용
getProperty("ball", _x);         // 상대 레퍼런스를 문자열로 표현함
getProperty("_root.ball", _x);   // 문자열로 점 연산자를 이용한 점
                                // 경로를 표현함
getProperty("/ball", _x);        // 슬래시를 이용한 경로를 문자열로
                                // 표현함

```

다음과 같이 점 연산자나 중괄호 연산자를 이용하여 속성을 액세스하는 것이 더 편리하다.

```

ball._x;
_root.ball._x;
ball["_x"];
_root["ball"]["_x"];

```

참조

setProperty(); 13장의 '무비 클립의 객체성'

getTimer() 전역 함수

무비가 실행된 이후로 소요된 시간을
밀리초 단위로 리턴하는 함수

버전 플래시 4 이후

문법 getTimer()

리턴 값

무비가 재생되기 시작한 이후로 경과한 시간을 밀리초 단위로 나타낸 값

설명

getTimer() 함수는 무비가 시작된 뒤 경과한 시간을 밀리초 단위로 나타낸 값을 구하는 함수이다. getTimer() 함수를 여러 번 이용하면 어떤 코드 블록을 정확한 시간 간격으로 실행할 수 있고, 게임 같은 것을 만들 때 카운트다운을 하는 것처럼

시간을 기반으로 하는 무비 기능을 구현할 수도 있다. 예를 들어 counterOutput이라는 텍스트 필드가 있는 무비 클립에서 다음과 같은 코드를 실행하면 60부터 0까지 카운트다운을 할 수 있다.

```
onClipEvent (load) {
    // 현재 시각 기록
    var startTime = getTimer();
    // 몇 초 동안 카운트다운할지 설정함
    var countAmount = 60;
    // 경과 시간을 저장해 둘 변수 초기화
    var elapsed = 0;
}

onClipEvent (enterFrame) {
    // 경과 시간 확인
    elapsed = getTimer() - startTime;
    // 카운트다운할 시간보다 경과 시간이 짧으면
    if (elapsed < countAmount * 1000) {
        // 남은 시간을 텍스트 필드에 출력한다.
        counterOutput = countAmount - Math.floor(elapsed / 1000);
    } else {
        // 그렇지 않으면 사용자에게 카운트다운이 끝났음을 알린다.
        counterOutput = "Time's UP!";
    }
}
```

무비가 실행된지 몇 초가 지났는지 확인하고 싶다면, getTimer()의 리턴 값을 1000으로 나누고 Math.floor()(버림)나 Math.round()(반올림), Math.ceil()(올림) 등을 이용하여 소수점 이하의 값을 잘라내면 된다.

```
numSeconds = Math.floor(getTimer()/1000);
```

예제

다음과 같은 코드를 이용하면 10초 동안 두 개의 프레임 사이에서 루프를 돌리고 나서 무비를 재생시킨다.

```
now = getTimer();
if (now > 10000) {
    play();
} else {
```

```
gotoAndPlay(_currentframe - 1);
}
```

참조

Date(), Date 클래스

getUrl() 전역 함수

브라우저로 문서를 가져오거나 서버 측 스크립트를 실행하거나 매크로미디어 디렉터 이벤트를 구동시키는 함수

버전 플래시 2, 플래시 3(플래시 4에서는 method 매개변수가 추가됨), 플래시 5

문법

```
getUrl (URL)
getUrl (URL, window)
getUrl (URL, window, method)
```

인자

URL 가져올 문서나 실행시킬 외부 스크립트의 절대 경로 또는 상대 경로를 나타내는 문자열

window 문서를 가져올 창이나 프레임의 이름을 지정하기 위한 문자열. 사용자가 정한 이름을 사용해도 되고 미리 정해진 이름인 “_blank”, “_parent”, “_self”, “_top”을 사용해도 된다(옵션).

method 현재 타임라인의 변수를 외부 스크립트로 보내는 방법(“GET” 또는 “POST”)을 지정하기 위한 문자열. 이 매개변수에는 어떤 변수나 표현식을 사용할 수 없고 반드시 리터럴 문자열을 사용해야 한다. 독립형 플래시 플레이어에서는 method 인자에 어떤 값을 지정하더라도 “GET” 메소드만 사용한다(옵션).

설명

getUrl() 함수는 다음과 같은 경우에 사용한다.

- 웹 브라우저의 프레임이나 창에 문서(보통은 웹 페이지)를 가져올 때

- 서버측 스크립트를 실행하고 그 결과를 브라우저 프레임이나 창으로 가져올 때
- 웹 브라우저에서 자바스크립트를 실행시킬 때
- 디렉터에 스프라이트(sprite)로 들어간 플래시 자원의 이벤트를 구동시킬 때

현재 창이나 프레임에 문서를 가져올 때는 window나 method 인자를 지정하지 않고 문서의 URL만 적어주면 된다. 플래시에서는 절대 URL("http:")와 같은 프로토콜 뒤에 서버 이름과 디바이스 이름이나 디바이스 이름을 적어준 것)과 상대 URL(현재 위치를 기준으로 하는 URL)을 모두 지원한다.

```
getURL("http://www.moock.org/");           // 웹 페이지의 절대 URL
getURL("file:///C:/WINDOWS/Desktop/index.html"); // 로컬 파일에 대한 절대 URL
getURL("/whatever/index.html");           // 상대 URL (http 프로토콜)
```

이름이 있는 창이나 프레임에 문서를 가져올 경우에는 window 인자에 창이나 프레임의 이름을 적어주면 된다.

```
getURL("http://www.moock.org/", "contentFrame"); // 프레임에 로딩할 때
getURL("http://www.moock.org/", "remoteWin");    // 창으로 로딩할 때
```

현재 무비가 포함된 프레임셋으로 문서를 불러올 때는 window 인자에 "_parent"라고 적어주면 된다.

```
getURL("http://www.moock.org/", "_parent");
```

웹 페이지의 모든 프레임셋을 새로 가져오는 문서로 채울 때는 "_top"을 window 인자로 전달하면 된다.

```
getURL("http://www.moock.org/", "_top");
```

문서를 새로운 창에 띄우려면 window 인자에 "_blank"를 전달한다.

```
getURL("http://www.moock.org/", "_blank");
```

“_blank”를 이용하여 새로운 창에서 문서를 열면 새로운 창의 외형(크기, 도구모음 설정, 위치 등)을 조절할 수 없다는 점에 주의하자. `getURL()`을 이용하여 사용자가 원하는 형태의 창을 열고 싶다면 무비의 호스트 페이지에 있는 자바스크립트 함수를 호출해야 한다. 자바스크립트를 이용하여 창을 띄우는 방법은 <http://www.moock.org/webdesign/flash>에 나와 있다.

`getURL()` 함수를 이용하여 원격 서버 애플리케이션이나 스크립트에 변수를 전달할 수도 있다. 현재 무비 클립의 타임라인 변수를 외부 스크립트로 보낼 때는 URL 인자에 스크립트의 이름을 적어주고 method 인자를 “GET”이나 “POST”로 설정하면 된다.

```
getURL("http://www.someserver.com/cgi-bin/search.pl", "resultsFrame",
"GET");
```

`getURL()`을 무비 클립 메소드로 사용하면 그 클립의 타임라인 변수를 전달할 수도 있다.

```
// myClip의 변수를 search.pl로 보낸다.
myClip.getURL("http://www.server.com/cgi-bin/search.pl",
"resultsFrame", "GET");
```

스크립트를 실행한 결과는 `getURL()`의 window 인자로 지정한 창이나 프레임에 출력된다(변수를 보낼 때는 window 인자를 반드시 지정해야 한다).

스크립트를 실행한 결과를 현재 프레임이나 창에서 열고 싶다면 window 인자의 값으로 “_self”를 사용하면 된다.

```
getURL("http://www.someserver.com/cgi-bin/search.pl", "_self", "GET");
```

method 인자를 “GET”으로 설정하면 현재 무비 클립 타임라인의 변수가 HTTP GET 요청의 스크립트 URL에 ‘질의 문자열(query string)’이 붙어서 전달된다. 질의 문자열은 변수 이름/값의 쌍으로 구성되며 각 변수는 & 기호로 구분된다.

```
http://www.someserver.com/cgi-bin/search.pl?term=javascript&scope
=entiresite
```

method 인자를 “POST”로 설정하면 현재 무비 클립 타임라인 변수를 HTTP POST 요청 헤더 뒤에 별도의 데이터 블록으로 전달한다(POST 메소드를 이용하는 HTML 폼을 사용하는 경우와 똑같다). 독립형 플래시 플레이어에서는 “POST” 메소드를 사용할 수 없다.

대부분의 웹 서버에서는 URL의 길이를 255자에서 1024자 이내로 제한하기 때문에, 용량이 큰 데이터를 전송할 때는 “GET” 대신 “POST”를 사용해야 한다.

getURL()에서 호출한 스크립트에서 보내는 정보는 플래시가 아닌 웹 브라우저에 표시된다는 점에 주의하자. 스크립트 실행 결과를 플래시에서 처리할 때는 loadVariables()를 사용해야 한다.

getURL() 함수는 [표 R-6]에 나온 것처럼 ‘http:’ 외에 다른 여러 프로토콜에서도 사용할 수도 있다.

[표 R-6] getURL에서 사용할 수 있는 프로토콜

프로토콜	형식	용도
event	“event: eventName params”	플래시 자원을 디렉터의 스프라이트로 사용하는 경우 디렉터에 이벤트를 전달할 때
file	“file:///driveSpec/folder/filename”	로컬 파일을 사용할 때
ftp	“ftp://server.domain.com/folder/filename”	FTP를 통해 원격 파일을 사용할 때
http	“http://server.domain.com/folder/filename”	HTTP를 통해 원격 파일을 사용할 때
javascript	“javascript: command”	브라우저의 자바스크립트 명령을 실행할 때
lingo	“lingo: comand”	플래시 자원을 디렉터의 스프라이트로 사용하는 경우 링고 명령을 실행할 때
print	“print:”, “targetClip”	플래시 5에서 print() 함수가 도입되기 전, 플래시 4에서 인쇄할 때 사용한 기능
vbscript	“vbscript: command”	브라우저의 VB스크립트 명령을 실행할 때

[표 R-6]에 나온 것처럼 플래시 자원을 매크로미디어 디렉터 파일에 집어넣으면, `getURL()`을 이용하여 링고 이벤트를 발생시키거나 링고 명령을 실행시킬 수 있다(링고는 액션스크립트와 유사한 디렉터의 스크립트 언어이다). 예를 들어 다음과 같은 형식의 프레임 이벤트를 추가할 수도 있다.

```
getURL ("event: eventName params");
```

이렇게 하면 디렉터에서 on eventName이라는 링고 이벤트 핸들러가 호출된다. 아래 코드에서는 `getURL()` 선언문을 이용하여 “myEvent”라는 이벤트를 만들어 그 이벤트에 “A”라는 문자열을 전달한다. “기호는 \”와 같이 이스케이프 시퀀스를 이용하여 표시한다는 점에 주의하자.

```
getURL ("event: myEvent \"A\""); // 디렉터에 이벤트를 보낸다.
```

이벤트를 받아들이기 위해 디렉터에 추가하는 플래시 스프라이트 자원에 추가할 링고의 스프라이트 이벤트 핸들러는 다음과 같다. 디렉터 스프라이트는 플래시 무비 클립 인스턴스와 거의 비슷하다. 링고의 `put` 키워드는 액션스크립트의 `trace()` 명령어와 비슷하고, `&&`는 링고에서 문자열을 합치는 데 사용하는 연산자이다.

```
on myEvent msg
    put "The message received from Flash was " && msg
end
```

다음과 같이 “lingo:” 키워드를 이용하여 디렉터에 있는 플래시 스프라이트에서 링고를 실행시킬 수도 있다.

```
getURL ("lingo: beep"); // 디렉터에서 뽀 소리가 나도록 한다.
```

디렉터 8.0에서는 플래시 5의 .swf 파일을 불러올 수 없지만, 이 책이 나올 무렵이면 업데이트된 플래시 자원 Xtra가 나와 있을 것이다.

`getURL()`을 이용하여 자바스크립트 코드를 실행시킬 수도 있다. `getURL()` 함수를 이용하여 다음과 같이 자바스크립트의 `alert` 명령어를 실행시킬 수도 있다.

```
getURL ("javascript: alert('hello world');");
```

자바스크립트 코드를 URL을 통해 실행시키는 방법은 매킨토시용 인터넷 익스플로러 4.5에서는 사용할 수 없다.

예제

웹 페이지로 연결되는 버튼에서는 일반적으로 다음과 같은 코드를 사용한다.

```
on (release) {  
    getURL("http://www.macromedia.com/");  
}
```

버그

매킨토시용 인터넷 익스플로러 4.5 이전 버전에서는 getURL()을 호출할 때 “POST” 메소드를 사용할 수 없다. 이러한 브라우저에서도 사용하려면 “POST” 대신 “GET”을 사용해야 한다(“GET”을 사용하면 앞에서 언급한 것처럼 데이터 용량에 제한이 있다).

대부분의 브라우저에서는 getURL()의 상대 링크를 .swf 파일을 포함하고 있는 HTML의 경로를 기준으로 해석한다. 하지만 매킨토시용 IE 4.5 이전 버전에서는 HTML 파일이 아니라 .swf 파일을 기준으로 해석한다. 따라서 .swf 파일과 HTML 파일이 서로 다른 디렉토리에 있는 경우에는 문제가 생길 수도 있다. 이러한 문제를 해결하려면 .swf 파일과 .html 파일을 같은 디렉토리에 넣거나 getURL()을 호출할 때 다음과 같이 절대경로를 사용해야 한다.

```
getURL ("http://www.someserver.com/")
```

참조

loadVariables(), MovieClip.getURL(), MovieClip.loadVariables(): ‘18장. 온스크린 텍스트 필드’의 <A>(앵커 또는 하이퍼텍스트 링크)

getVersion() 전역 함수

플래시 플레이어 버전과 컴퓨터 플랫폼을
확인하는 함수

버전 플래시 5

문법 getVersion()

리턴 값

무비가 실행되고 있는 플래시 플레이어의 버전 및 플랫폼에 대한 정보를 포함하고 있는 문자열

설명

getVersion() 함수를 이용하면 무비를 실행하고 있는 플랫폼과 플래시 플레이어 버전을 알 수 있다. 운영체제나 플래시 플레이어 버전에 따라 다른 코드를 실행시키 고자 할 때 이 함수를 이용한다. getVersion()에서 리턴된 문자열의 형식은 다음과 같다.

```
platform majorVersion,minorVersion,buildNumber,patch
```

platform은 플랫폼을 나타내는 코드("WIN", "MAC", "UNIX")이며, 그 뒤에는 메 이저 버전, 마이너 버전, 빌드(개정판) 번호가 들어간다. 마지막의 patch는 보통 0 이다.

```
WIN 5,0,30,0 // 윈도우, 버전 5.0, 빌드 30 (5.0r30)
MAC 5,0,41,0 // 매킨토시, 버전 5.0, 빌드 41(5.0r41)
UNIX 4,0,12,0 // 유닉스, 버전 4.0, 빌드 12(4.0r12)
```

매크로미디어 문서에 나와있는 것과는 달리 getVersion() 함수는 플래시 저작 도구의 Test Movie 모드에서도 제대로 작동한다. 그 경우에는 플래시 저작 도구에 내장된 플래시 플레이어의 버전 번호(저작 도구 자체의 버전 번호와는 다를 수 있다) 가 출력된다. 예를 들어 플래시 5 저작 도구에 내장된 플레이어 버전은 5.0r30이므 로, getVersion() 함수를 실행시키면 다음과 같이 출력된다.

```
WIN 5,0,30,0
또는
MAC 5,0,30,0
```

저작 도구의 메이저 버전이나 마이너 버전을 만들 때마다 buildNumber는 0부터 다시 시작한다. 하지만 일반적인 플래시 저작 도구의 개발 사이클을 보면 저작 도구 의 최종 버전이 나오기 전에 보통 플래시 플레이어의 빌드 번호가 여러 번 바뀐다. 따라서 새로운 메이저 버전의 플레이어가 나올 때 빌드 번호가 0보다 큰 경우가 대 부분이다. 예를 들어 플래시 5 플레이어는 빌드 번호가 30번일 때 처음으로 공식 버 전이 발표되었다. 아마도 플래시 6 플레이어가 공식적으로 발표되면 버전이 다음과 같이 나올 것이다.

```
WIN 6,0,xx,0
```

하지만 플래시 6에서 만든 무비를 플래시 5 플레이어에서 실행하면 getVersion() 함수에서는 다음과 같은 문자열을 리턴할 것이다.

```
WIN 5,0,30,0
```

보통 플레이어의 플랫폼, 메이전 버전, 빌드 번호가 중요하기 때문에, '4장. 원시 데이터형'에서 배운 문자열 조작 도구를 이용하여 getVersion()에서 리턴된 문자열 중에서 필요한 부분만 뽑아내거나 각 성분을 속성으로 저장할 수 있도록 사용자 정의 객체를 만드는 것이 좋다.

플래시 플레이어 버전에 따라 다른 액션스크립트 코드를 사용하는 것이 아니라면 버전을 알아내거나 브라우저 정보를 알아내거나 자동으로 페이지를 전환할 때 자바스크립트나 VB스크립트를 이용하는 것이 더 좋다. 또한 사용 중인 플래시 플레이어 버전이 5.0 이후 버전이 아니면 getVersion()을 사용할 수도 없다. 자바스크립트와 VB스크립트를 이용하여 플래시 플레이어 설치 여부나 버전 번호를 알아내는 방법은 <http://www.moock.org/webdesign/flash/detection/mookfpi>에 나와 있다.

예제

아래 코드는 getVersion()에서 리턴된 문자열의 각 부분을 추출하여 객체의 속성으로 저장함으로써 손쉽게 액세스할 수 있도록 해주는 예제이다.

```
// getVersion()에서 리턴된 문자열에서 필요한 내용을 추출한다.
var version = getVersion();
var firstSpace = version.indexOf(" ");
var tempString = version.substring(firstSpace + 1, version.length);
var tempArray = tempString.split(",");

// 각 부분을 객체에 저장한다.
// 버전 번호 부분은 정수로 저장한다.
var thePlayer = new Object();
thePlayer.platform = version.substring(0, firstSpace);
thePlayer.majorVersion = parseInt(tempArray[0]);
thePlayer.minorVersion = parseInt(tempArray[1]);
thePlayer.build = parseInt(tempArray[2]);
thePlayer.patch = parseInt(tempArray[3]);

// 객체를 활용하여 버전에 따라 다른 코드를 실행시킬 수 있다.
if (thePlayer.platform == "WIN") {
```

```
// 윈도우에서만 사용하는 코드
} else if (thePlayer.platform == "MAC") {
    // 매킨토시에서만 사용하는 코드
} else if (thePlayer.platform == "UNIX") {
    // 유닉스에서만 사용하는 코드
}

if ((thePlayer.majorVersion == 5) && (thePlayer.build == 30)) {
    trace ("I recommend upgrading your player to avoid text display
    bugs");
}
```

참조

\$version; '부록 C. 하위 호환성'

gotoAndPlay() 전역 함수

플레이헤드를 주어진 프레임으로 옮기고
무비나 클립을 재생한다.

버전 플래시 2 이후

문법

```
gotoAndPlay(frameNumber)
gotoAndPlay(frameLabel)
gotoAndPlay(scene, frameNumber)
gotoAndPlay(scene, frameLabel)
```

인자

frameNumber

무비나 클립을 재생하기 전에 현재 타임라인의 플레이헤드를 이동시킬 프레임 번호를 나타내는 양의 정수. *frameNumber*가 1보다 작거나 타임라인에 있는 프레임 수보다 크면 플레이헤드를 각각 첫 프레임 또는 마지막 프레임으로 이동시킨다.

frameLabel

현재 타임라인의 플레이헤드를 이동시킬 프레임 레이블을 나타내는 문자열. *frameLabel*이 없으면 플레이헤드를 타임라인의 첫 번째 프레임으로 보낸다.

scene 주어진 `frameNumber`나 `frameLabel`을 포함하고 있는 장면(scene)의 이름을 나타내는 문자열. 이 값이 없으면 현재 장면을 사용한다(옵션).

설명

`gotoAndPlay()` 함수를 호출할 때 `scene` 인자를 전달하지 않으면, 현재 타임라인의 플레이헤드가 `frameNumber` 또는 `frameLabel`로 지정한 프레임으로 움직이고 그 위치부터 타임라인을 재생한다. '현재 타임라인'은 `gotoAndPlay()` 함수를 호출한 무비 클립이나 무비를 의미한다.

`gotoAndPlay()` 함수를 호출할 때 두 개의 인자를 사용하면 첫 번째 인자는 `scene`으로 간주된다. 한 개의 인자를 사용하면 `frameNumber`나 `frameLabel`로 간주되며 현재 장면을 기준으로 설정된다.

`gotoAndPlay()` 함수를 호출할 때 `scene` 인자를 사용하면 플레이헤드가 주어진 장면에서 `frameNumber`나 `frameLabel`로 지정한 프레임으로 이동하며 그 장면에서 재생이 시작된다. `scene` 인자를 사용할 때는 `gotoAndPlay()` 함수를 `_root` 타임라인에서 호출해야 한다. 다른 타임라인에서 `gotoAndPlay()` 함수를 호출할 때 `scene` 인자를 사용하면, 아무런 오류 메시지 없이 함수가 종료되며 플레이헤드가 다른 곳으로 움직이지 않는다. 무비를 재생할 때는 여러 장면이 합쳐져 하나의 타임라인이 된다. 즉 1번 장면의 타임라인에 20개의 프레임이 있고 2번 장면의 타임라인에 10개의 프레임이 있으면, `gotoAndPlay(25)`라는 선언문을 사용하면 2번 장면의 5번 프레임으로 플레이헤드가 움직인다.



액션스크립트를 많이 사용하는 무비에서는 장면을 사용하지 않는 것이 좋다. 무비 클립과는 달리 장면은 객체로 표현되지 않기 때문에 내장 함수를 이용하여 직접 조작할 수 없다. 플래시의 장면 기능을 이용하는 것보다는 레이블과 무비 클립을 타임라인의 장면과 비슷하게 사용하는 것이 낫다.

전역 함수인 `gotoAndPlay()` 함수에서는 현재 무비 클립에만 영향을 미친다. 같은 타임라인에 있는 다른 무비 클립의 프레임이나 상태는 바뀌지 않는다. 다른 무비 클립을 재생하려면 각 무비 클립별로 `play()`나 `gotoAndPlay()` 명령을 따로 실행시켜야 한다. 현재 무비 클립이 아닌 다른 클립에 대해 `gotoAndPlay()` 함수를 적용하려면, `myClip.gotoAndPlay()`와 같은 무비 클립 메소드를 이용하면 된다.

버그

플래시 플레이어의 빌드 5.0r30에서는 onClipEvent() 핸들러에서 gotoAndPlay() 함수를 호출할 때 frameLabel에 문자열 리터럴을 사용하면 함수가 제대로 작동하지 않는다. 이 버그를 해결하려면 gotoAndPlay("myLabel") 대신 this.gotoAndPlay("myLabel")과 같이 this를 이용하여 현재 클립의 무비 클립 메소드 형태로 호출하면 된다.

예제

```
// 현재 타임라인의 5번 프레임으로 움직여서 무비를 재생시킨다.
gotoAndPlay(5);
// exitSequence 장면의 10번 프레임으로 움직여서 무비를 재생시킨다.
gotoAndPlay("exitSequence", 10);
// exitSequence 장면의 "goodbye" 프레임으로 움직여서 무비를 재생시킨다.
gotoAndPlay("exitSequence", "goodbye");
// 이렇게 하면 현재 장면의 "exitSequence"라는 프레임으로 움직인다.
gotoAndPlay("exitSequence");
// exitSequence 장면의 1번 프레임을 재생시킨다.
gotoAndPlay("exitSequence", 1);
```

참조

gotoAndStop(), MovieClip.gotoAndPlay(), play(), stop()

gotoAndStop() 전역 함수

플레이헤드를 주어진 프레임으로 이동시키고
현재 클립을 정지시키는 함수

버전 플래시 2 이후

문법

```
gotoAndStop(frameNumber)
gotoAndStop(frameLabel)
gotoAndStop(scene, frameNumber)
gotoAndStop(scene, frameLabel)
```

인자

frameNumber

현재 타임라인의 플레이헤드를 옮길 프레임 번호를 나타내는 양의 정수. *frameNumber*가 1보다 작거나 타임라인에 있는 프레임 개수보다 더 크면 플레이헤드는 첫 번째 프레임 또는 마지막 프레임으로 움직인다.

frameLabel

현재 타임라인의 플레이헤드를 옮길 프레임 레이블을 나타내는 문자열. *frameLabel*이 없으면 플레이헤드는 타임라인의 첫 번째 프레임으로 움직인다.

scene

주어진 *frameNumber*나 *frameLabel*을 포함하는 장면의 이름을 나타내는 문자열. 이 인자를 전달하지 않으면 현재 장면을 기준으로 한다(옵션).

설명

`gotoAndStop()`을 호출할 때 *scene* 인자를 전달하지 않으면 현재 타임라인의 플레이헤드를 *frameNumber*나 *frameLabel* 인자로 지정한 프레임으로 이동시킨다. '현재 타임라인'은 `gotoAndStop()` 함수를 호출한 무비 클립이나 무비를 의미한다. 플레이헤드는 그 프레임에서 멈추며 대상 프레임으로 이동한 후에 자동으로 다음 프레임으로 움직이지 않는다.

`gotoAndPlay()` 함수를 호출할 때 두 개의 인자를 사용하면 첫 번째 인자는 *scene*으로 간주된다. 한 개의 인자만 사용하면 그 인자는 *frameNumber* 또는 *frameLabel*로 처리되며 현재 장면을 기준으로 한다.

`gotoAndStop()` 함수를 호출할 때 *scene* 인자를 사용하면 그 인자로 지정한 장면 안에 있는 프레임으로 플레이헤드가 움직인다. *scene* 인자를 사용하려면 `gotoAndStop()` 함수를 반드시 `_root` 타임라인에서 호출해야 한다. 다른 타임라인에서 호출할 때 *scene* 인자를 사용하면 아무런 오류 메시지 없이 함수가 종료되고 주어진 프레임으로 플레이헤드가 움직이지 않는다.

`gotoAndStop()` 함수는 현재 무비 클립에만 영향을 미친다. 같은 타임라인에 있는 다른 무비 클립의 프레임이나 상태는 바뀌지 않는다. 다른 무비 클립의 플레이헤드를 이동시키려면 각 무비 클립별로 `gotoAndStop()` 함수를 따로 호출해야 한다.

현재 무비 클립이 아닌 다른 클립에 gotoAndStop() 함수를 적용하려면 무비 클립 메소드 형태인 myClip.gotoAndStop()을 사용해야 한다.

버그

플래시 플레이어의 빌드 5.0r30에서는 onClipEvent() 핸들러에서 gotoAndStop() 함수를 호출할 때 frameLabel에 문자열 리터럴을 사용하면 함수가 제대로 작동하지 않는다. 이 버그를 해결하려면 gotoAndStop("myLabel") 대신 this.gotoAndStop("myLabel")과 같이 this를 이용하여 현재 클립의 무비 클립 메소드 형태로 호출하면 된다.

예제

```
// 현재 타임라인의 5번 프레임으로 움직이고 무비를 중지한다.
gotoAndStop(5);
// introSequence 장면의 20번 프레임으로 가서 멈춘다.
gotoAndStop("introSequence", 20);
// introSequence의 "hello" 프레임으로 가서 멈춘다.
gotoAndStop("introSequence", "hello")
// 이렇게 하면 현재 장면의 "introSequence" 프레임으로 간다.
gotoAndStop("introSequence")
// introSequence 장면의 1번 프레임으로 간다.
gotoAndStop("introSequence", 1)
```

참조

gotoAndPlay(), MovieClip.gotoAndStop(), play(), stop()

_highquality 전역 속성

플레이어의 렌더링 화질

버전 플래시 4(플래시 5에서는 _quality를 쓰는 것이 좋다)

문법 _highquality

액세스 읽기/쓰기

설명

_highquality 전역 속성에는 플래시 플레이어의 렌더링 화질을 조절하기 위한 0과 2 사이의 정수가 저장된다.

- 0 저화질. 비트맵과 벡터에 모두 안티앨리어싱(테두리를 부드럽게 만드는 것)을 적용하지 않는다.
- 1 고화질. 벡터에만 안티앨리어싱이 적용된다. 비트맵에는 애니메이션이 아닌 경우에만 안티앨리어싱이 적용된다.
- 2 최고화질. 비트맵과 벡터에 모두 안티앨리어싱이 적용된다.

플래시 5에서는 `_highquality` 대신 `_quality` 속성을 이용한다. `_quality`에서는 무비의 화질을 “Low”, “Medium”, “High”, “Best”로 설정할 수 있다.

참조

`_quality`, `toggleHighQuality()`

#include 지시자

외부 액션스크립트 파일의 텍스트를 가져오기 위한 지시자

버전 플래시 5

문법 `#include path`

인자

path 외부에서 불러올 스크립트 파일의 위치와 이름을 나타내는 문자열. 위치를 지정할 때는 `.fla` 파일에 대해 상대 경로 또는 절대 경로를 모두 사용할 수 있다(예제 참조). 경로를 지정할 때는 역슬래시(\)가 아니라 그냥 슬래시(/)를 사용해야 한다는 점에 주의하자. 스크립트 파일은 확장자를 `.as`로 저장해야 한다.

설명

`#include` 지시자를 이용하면 외부 텍스트 파일(`.as` 확장자로 저장하는 것이 좋다)로부터 스크립트 텍스트를 불러와서 실행 중인 스크립트의 `#include` 지시자가 있는 자리에 집어넣을 수 있다. `#include` 작업은 컴파일할 때 이루어지므로 무비에 포함될 텍스트는 저작 도구에서 무비를 테스트하고 `.swf` 파일로 저장할 때 주어진 경로에 있어야 한다. 무비를 `.swf` 파일로 저장한 다음 외부 파일을 고치면 그 변경 사항이 무비에는 반영되지 않는다. 따라서 외부 스크립트 파일을 수정했을 때 그 무비에도 그 결과를 반영하려면 무비를 다시 `.swf` 파일로 저장해야 한다.

#include 지시자는 어떤 코드 블록을 여러 스크립트에 집어넣거나 플래시 프로젝트 전반에 걸쳐서 사용할 때 유용하다(외부 자원 라이브러리와 용도가 비슷하다). 보통 코드를 집중시킬 때, 버전 제어 시스템 도구(CVS나 Microsoft Visual Source Safe 등)에서 코드를 관리하거나 액션스크립트 에디터보다 쓰기 좋은 외부 텍스트 에디터를 사용할 때 #include 지시자를 많이 사용한다. 또한 플래시 애니메이션을 그래픽 전문가가 따로 만들고 프로그래머는 코딩만 하는 경우에도 이러한 방법을 사용하는 것이 좋다. 현재 타임라인이나 무비 클립과는 무관한 함수 라이브러리 같은 코드 저장고로 외부 파일을 사용하는 것도 좋은 방법이다. 플래시 파일과 밀접하게 연결된 코드는 외부 파일에 저장하는 것이 그다지 효율적이지 못하다.

주의 사항

#include 지시자는 # 기호로 시작되며 괄호를 사용하지도 않고 세미콜론으로 끝나지 않는다는 점에 주의하자. #include 문 뒤에 세미콜론을 붙이면 오류가 생기면서 외부 파일을 불러올 수 없게 된다. 주어진 경로에 파일이 없어도 오류가 발생하고 외부 파일이 스크립트에 추가되지 않는다. 액션 패널 메뉴(패널의 오른쪽 위 화살표 버튼에 있는 메뉴)에서 Check Syntax 명령(Ctrl-T 또는 Command-T)을 이용하여 문법 검사를 할 때는 외부 파일에 있는 텍스트도 검사한다.

예제

아래 코드를 이용하면 myScript.as라는 외부 파일을 현재 편집 중인 .fla 파일로 불러올 수 있다. 다음과 같은 상대 경로를 사용할 때는 #include 지시자가 있는 .fla 파일과 같은 폴더에 myScript.as 파일을 저장해야 한다.

```
#include "myScript.as"
```

상대 경로에 하위 디렉토리를 포함시킬 수도 있다. 아래와 같은 코드는 myScript.as가 .fla 파일보다 한 레벨 아래인 includes라는 하위 디렉토리에 있는 경우에 사용할 수 있다.

```
#include "includes/myScript.as"
```

현재 폴더 바로 위 폴더는 두 개의 점(..)으로 표시한다. 다음과 같은 코드를 이용하여 .fla 파일보다 한 단계 위에 있는 myScript.as 파일을 불러올 수 있다.

```
#include "../myScript.as"
```

.fla 파일이 저장된 폴더의 상위 디렉토리에 있는 includes라는 디렉토리에 myScript.as 파일이 있다면 다음과 같은 식으로 불러올 수 있다.

```
#include "../includes/myScript.as"
```

다음과 같이 임의의 폴더를 절대 경로로 지정할 수도 있다.

```
#include "C:/WINDOWS/Desktop/myScript.as"
```

하지만 절대 경로를 사용하면 플랫폼에 따라 내용이 달라지기 때문에, .fla 파일을 다른 시스템의 다른 디렉토리에서 컴파일할 때는 그 부분을 수정해야 한다.

```
#include "C:/WINDOWS/Desktop/myScript.as"           // 윈도우
#include "Mac HD:Desktop folder:working:myScript.as" // 매킨토시
```

참조

16장의 '액션스크립트 코드 외부화' 참조

Infinity 전역 속성

무한대를 나타내는 상수

버전	플래시 5
문법	Infinity
액세스	읽기 전용

설명

액션스크립트에서 표현할 수 있는 가장 큰 숫자보다 큰 모든 숫자는 Infinity라는 숫자 상수로 표현된다. 액션스크립트에서 사용할 수 있는 가장 큰 수는 Number.MAX_VALUE(1.7976931348623157e+308)로 표현된다.

예제

계산 결과가 액션스크립트에서 사용할 수 있는 가장 큰 숫자보다 더 크면, 그 리턴 값은 Infinity가 된다.

```
Number.MAX_VALUE * 2;    // Infinity
```

양수를 0으로 나눈 결과도 Infinity가 된다.

```
1000 / 0;           // Infinity
```

주의 사항

Infinity는 Number.POSITIVE_INFINITY를 줄여 쓴 상수이다.

참조

-Infinity, Number.POSITIVE_INFINITY; 4장의 '숫자 데이터형에 있는 특수 값'

-Infinity 전역 속성

음의 무한대를 나타내는 상수

버전	플래시 5
문법	-Infinity
액세스	읽기 전용

설명

액션스크립트에서 음수로 사용할 수 있는 가장 작은 숫자보다 작은 수는 -Infinity라는 숫자 상수로 표현된다. 액션스크립트에서 사용할 수 있는 가장 작은 음수(즉 절대값이 가장 큰 음수)는 -Number.MAX_VALUE이며, 그 값은 -1.797693 1348623157e+308과 같다.

예제

계산 결과가 액션스크립트에서 사용할 수 있는 가장 작은 숫자보다 더 작으면 그 리턴 값은 -Infinity가 된다.

```
-Number.MAX_VALUE * 2;    // -Infinity
```

음수를 0으로 나눈 결과도 -Infinity가 된다.

```
-1000 / 0;               // -Infinity
```

주의 사항

-Infinity는 Number.NEGATIVE_INFINITY를 줄여 쓴 상수이다.

참조

Infinity, Number.NEGATIVE_INFINITY; 4장의 '숫자 데이터형에 있는 특수 값'

int() 전역 함수

소수점 이하 부분을 잘라내는 함수

버전 플래시 4(플래시 5에서는 Math 클래스의 메소드를 사용하는 것이 좋다)

문법 `int(number)`

인자

number 숫자 또는 숫자 값을 가지는 표현식. 일반적으로 소수점 이하 부분이 있는 숫자를 사용한다.

리턴 값

number의 정수 부분

설명

int() 함수는 플래시 4에서 숫자의 정수 부분을 구하기 위해 사용하는 함수이다. 이 함수에서는 양수에 대해서는 내림을, 음수에 대해서는 올림을 적용한다. int() 함수는 -2147483648 (-231) 이상 2147483647(231-1) 이하의 값에만 사용할 수 있으며, 그 범위를 벗어나는 숫자를 사용하면 엉뚱한 결과가 나온다. number 인자가 숫자만으로 이루어진 문자열인 경우에는 int() 함수에서 자동으로 그 문자열을 숫자로 변환해서 처리한다. number가 true 값을 가지는 부울형 변수이면 1을 리턴한다. 다른 모든 숫자가 아닌 데이터(undefined 및 null 포함)를 인자로 사용하면 0을 리턴한다.

주의 사항

int() 함수는 더 이상 쓰이지 않으며 Math.floor(), Math.ceil(), Math.round() 메소드를 사용하는 것이 좋다. 숫자가 아닌 데이터를 정수 또는 숫자로 변환할 때는 parseInt()나 Number()를 사용하면 된다.

예제

```
int(4.5)      // 4
int(-4.5)     // -4
int(3.999)    // 3
```

어떤 숫자와 그 숫자를 `int()` 함수에 전달했을 때 리턴 값을 비교하면 그 숫자가 정수인지 확인할 수 있다.

```
if (int(x) != x) {
    trace ("Please enter a whole number for your age in years");
}
```

참조

`Math.ceil()`, `Math.floor()`, `Math.round()`, `Number()`, `parseFloat()`, `parseInt()`

isFinite() 전역 함수

어떤 숫자가 Infinity보다 작고 -Infinity보다 큰지
확인하는 함수

버전 플래시 5

문법 `isFinite(number)`

인자

number 숫자 값 또는 숫자 표현식

리턴 값

주어진 숫자가 `Number.MAX_VALUE` 이상, `-Number.MAX_VALUE` 이하이면 `true`, 그렇지 않으면 `false`를 리턴한다. `number`가 숫자 데이터형이 아닌 경우에는 `isFinite` 함수를 실행하기 전에 `number`를 숫자형으로 변환한다.

설명

`isFinite()` 함수는 주어진 숫자가 액션스크립트에서 사용할 수 있는 숫자 범위에 속하는지 확인하는 함수이다. 숫자 데이터가 들어가야만 제대로 작동하는 코드를 실행시키기 전에 미리 `isFinite()`로 확인해 보는 것이 좋다.

예제

```

if (!isFinite(x * y)) {           // 주어진 숫자가 유한한 숫자인지 확인한다.
    trace ("The answer is too large to display. Try again.");
}

isFinite(-2342434);               // true
isFinite(Math.PI);               // true
isFinite(Number.MAX_VALUE * 2)   // false

```

참조

-Infinity, Infinity, isNaN(), Number.MAX_VALUE, Number.MIN_VALUE; 4장의 '숫자 데이터형의 특수 값'

isNaN() 전역 함수

Nan 값과 같은지 확인하는 함수

버전 플래시 5

문법 `isNaN(value)`

인자

value 확인할 표현식

리턴 값

*value*가 NaN이면 true, 그렇지 않으면 false를 리턴한다.

설명

어떤 값이 NaN인지 확인하는 함수. 일반적인 등치 검사 방법을 사용하면 두 값 모두 NaN에 해당한다고 하더라도 반드시 두 값이 같다는 결과가 나오지 않기 때문에 isNaN() 함수를 이용해야 한다. 예를 들어 다음과 같은 표현식 값은 false이다.

```
NaN == NaN;
```

isNaN()은 어떤 코드에 보통 수학적인 오류(0을 0으로 나누는 것과 같은 경우)가 있는지 확인할 때, 또는 어떤 값을 숫자로 변환하는 작업이 제대로 처리되었는지 확인할 때 사용한다. isNaN()에서는 주어진 표현식이 유효한 숫자 값을 가지지 않는 경우에 true를 리턴하기 때문에, isNaN()을 사용할 때는 논리 NOT(!) 연산자를 함

계 사용하는 경우가 흔하다(숫자가 아닌 값이 아닌 것은 숫자이다). 0/0은 NaN이지만 양수를 0으로 나눈 값은 Infinity, 음수를 0으로 나눈 값은 -Infinity라는 점에 주의하자.

예제

```
// x 값을 설정한다.
var x = "test123";
// x를 수학 표현식에서 사용하기 전에 숫자로 사용할 수 있는지 먼저 확인한다.
// 사용자가 입력한 텍스트 필드는 언제나 문자열 데이터이므로
// 이러한 방법이 매우 유용하다.
if (!isNaN(parseFloat(x))) {
    var y = parseFloat(x) * 2;
}
```

참조

isFinite(), NaN; 4장의 '숫자 데이터형의 특수 값'

Key 객체

키보드에 있는 키의 상태를 알아내기 위한 객체

버전 플래시 5

문법 *Key.property*
Key.methodName()

속성

[표 R-7]은 Key 객체에 있는 속성 목록이다.

[표 R-7] Key 객체의 키코드 속성

속성	키코드	속성	키코드
BACKSPACE	8	INSERT	45
CAPSLOCK	20	LEFT	37
CONTROL	17	PGDN	34
DELETEKEY	46	PGUP	33
DOWN	40	RIGHT	39
END	35	SHIFT	16

속성	키코드	속성	키코드
ENTER	13	SPACE	32
ESCAPE	27	TAB	9
HOME	36	UP	38

메소드

getAscii() 마지막으로 누른 키의 ASCII 값을 리턴하는 메소드

getCode() 마지막으로 누른 키의 키코드를 리턴하는 메소드

isDown() 특정 키가 눌러져 있는지 확인하는 메소드

isToggled() Num Lock, Caps Lock, Scroll Lock 키가 활성화되어 있는지 확인하는 메소드

설명

Key 객체는 현재 눌러져 있는 키 또는 마지막으로 누른 키를 알아내기 위해 쓰인다. 이 객체를 이용하면 화살표 키로 우주선을 조종하는 게임과 같은, 키보드로 제어하는 인터페이스를 만들 수 있다.

키보드 종류가 다양하기 때문에 키보드로 제어하는 인터페이스를 만드는 것이 그다지 간단하지는 않다. 하지만 적절한 방법을 활용하면 모든 사용자가 같은 방식으로 인터페이스를 조작할 수 있도록 만들 수 있다.

키보드 명령을 감지하는 데는 크게 두 가지 방법이 있다.

- **isDown()** 메소드를 이용하여 어떤 키가 눌러져 있는 상태인지 알아내는 방법. 비디오 게임과 같이 계속해서 키를 눌러야 하는 경우에는 이러한 방법을 사용하는 것이 좋다.
- **getCode()**와 **getAscii()** 메소드를 이용하여 마지막으로 누른 키를 알아내는 방법. 키를 눌렀을 때 특정 작업을 처리하는 키보드로 조작하는 인터페이스에서는 이러한 방법을 사용하는 것이 좋다. 보통 **keyDown** 이벤트 핸들러에서 이 방법을 사용하여 서로 다른 키를 구분한다. 마지막으로 누른 키를 쉬지 않고 계속해서 확인할 필요는 없다. 사실 이렇게 하면 키를 연속으로

누르지도 않았는데 같은 작업을 쓸데없이 반복하게 된다. 즉 keyDown 이벤트 핸들러는 키를 누를 때마다 한 번씩만 실행되므로 keyDown 이벤트 핸들러에서만 getCode()나 getAscii() 메소드를 사용하는 것이 좋다.

getCode()와 isDown()에서 사용하는 윈도우 가상 키코드(줄여서 '키코드'라고 부름)는 키에 인쇄된 기호가 아닌 키보드에 있는 물리적인 키를 나타내는 숫자이다. 키코드를 이용하면 서로 다른 운영체제에서 무비를 실행하거나 서로 다른 언어를 사용하는 키보드 또는 키 배치가 다른 키보드를 사용하더라도 키를 정확하게 구분할 수 있다.

대부분의 키보드에서 A부터 Z에 해당하는 키코드는 Latin 1 문자에서 대문자에 해당하는 코드 포인트(65-90)와 같은 값을 가진다. 마찬가지로 0부터 9까지의 키코드도 Latin 1에서의 코드 포인트 값과 같다(48-57). 다른 키의 키코드는 Latin 1 코드 포인트와 조금씩 다르다. 하지만 글자나 숫자가 아닌 키의 키코드도 Key 속성을 이용하여 사용할 수 있다. 예를 들어 위쪽 화살표의 키코드가 38이라는 것을 굳이 외우지 않아도 Key.UP 속성을 이용하면 된다. 아래 코드를 이용하면 위쪽 화살표가 지금 눌러져 있는지 확인할 수 있다.

```
if (Key.isDown(Key.UP)) {
    trace("The up arrow is being pressed");
}
```

문자나 숫자도 아니고 Key 속성으로 나타낼 수도 없는 키(F1, F2와 같은 기능 키 등)를 다룰 때는 다음과 같은 식으로 원하는 키의 키코드를 알아내기 위한 간단한 테스트용 무비를 만들어보는 것이 좋다.

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

키코드는 부록 B에 수록되어 있다.

참조

10장의 'keyUp' 및 'keyDown', '부록 B. Latin 1 문자 범주 및 키코드'

Key.getAscii() 메소드

마지막으로 누른 키의 ASCII 값을 리턴하는 메소드

버전 플래시 5**문법** `Key.getAscii()`**리턴 값**

마지막으로 누른 키의 ASCII 값을 나타내는 정수

설명

`getAscii()` 메소드에서는 마지막으로 누른 키의 ASCII 값을 나타내는 정수를 리턴한다. 모든 키에 ASCII 값이 할당되어 있는 것은 아니므로 `getAscii()` 메소드는 보통 Latin 1 문자 세트(서유럽어)에 있는 문자나 숫자를 감지할 때만 사용한다. `getCode()`와는 달리 대문자와 소문자를 구분한다. 하지만 메인 키보드에 있는 8 키와 숫자 키패드에 있는 8 키처럼 ASCII 값은 같지만 실제 키의 위치가 서로 다른 경우는 구별할 수 없다.

ASCII 값이 아닌 키보드의 물리적인 위치를 기준으로 특정 키를 눌렀는지 확인할 때(예: 게임을 할 때 다이아몬드와 같은 패턴으로 네 개의 키를 사용하는 경우)는 `getCode()`를 사용해야 한다.

예제

아래 예제에서는 행맨(Hangman) 단어 게임에서 어떤 키가 눌러져 있는지 알아내는 방법을 보여준다. 이 코드에서는 `keyDown` 이벤트 핸들러를 이용하여 어떤 키가 눌러져 있다는 것을 감지하고 나서 그 키를 사용자가 선택한 글자 목록에 추가한다.

```
onClipEvent (keyDown) {
    var lastKey = Key.getAscii();
    guessNum++;
    userGuesses[guessNum] = String.fromCharCode(lastKey);
}
```

참조`Key.getCode()`; 부록 B, 10장의 'keyDown'

Key.getCode() 메소드

마지막으로 누른 키의 키코드를 리턴하는 메소드

버전 플래시 5**문법** `Key.getCode()`**리턴 값**

마지막으로 누른 키의 키코드에 해당하는 정수

설명

`getCode()` 메소드에서는 마지막으로 누른 키의 키코드를 리턴한다. 키코드는 키보드에서 키의 물리적인 위치를 나타내는 임의의 정수이다. 윈도우가 아닌 운영 체제에서도 그 운영 체제의 키코드 시스템을 플래시에서 자동으로 윈도우 형태로 변환해 주므로, `getCode()` 메소드는 서로 다른 플랫폼에서도 자유롭게 사용할 수 있다. 똑같은 ASCII 값을 가지는 서로 다른 키를 구분할 때도 `getCode()`를 이용할 수 있다. 예를 들어 메인 키보드에 있는 8번 키와 숫자 키패드에 있는 8번 키를 `getAscii()` 메소드로는 구분할 수 없지만, `getCode()` 메소드를 사용하면 서로 다르다는 것을 알 수 있다. 하지만 `getCode()`에서는 대문자와 소문자를 구분할 수 없다 (예를 들어 A와 a는 같은 키에 할당되어 있기 때문에 키코드가 똑같다).

`Key` 객체에는 다양한 키의 키코드가 속성으로 저장되어 있다(`Key.UP`, `Key.BACKSPACE` 등). 특정 키의 키코드를 알고 싶다면 부록 B를 참조하거나 다음과 같은 식으로 키코드 테스트를 만들어서 확인할 수 있다.

1. 새로운 플래시 문서를 만든다.
2. 타임라인의 2번 프레임에 프레임을 추가한다.
3. 1번 프레임에 다음과 같은 코드를 추가한다.

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

4. Control → Test Movie를 선택한다.
5. 무비의 스테이지를 클릭한다.
6. 키를 누른다. 그 키의 키코드가 Output 창에 출력된다.

예제

isDown()과는 달리 getCode()는 각 키를 누를 때마다 매번 직접적인 결과가 나타나는 인터페이스를 만들 때 유용하다. 예를 들어 사용자가 스페이스바를 누르면 무비의 인트로 부분을 건너뛰도록 해보자. 사용자가 스페이스바를 누르면 플레이어헤드(메인 타임라인에 있는) 무비의 메인 인터페이스로 이동시키면 된다.

```
// intro 클립의 코드
onClipEvent (keyDown) {
    if (Key.getCode() == Key.SPACE) {
        _root.gotoAndStop("mainInterface");
    }
}
```

참조

Key.getAscii(), Key.isDown(); 부록 B, 10장의 'keyDown'

Key.isDown() 메소드

특정 키가 현재 눌러져 있는지 확인하는 메소드

버전 플래시 5

문법 Key.isDown(keycode)

인자

keycode 확인할 키의 키코드에 해당하는 정수. Key 객체에 포함된 상수(Key.UP, Key.BACKSPACE 등)를 이용해도 된다.

리턴 값

주어진 keycode에 해당하는 키가 눌러져 있으면 true, 그렇지 않으면 false를 리턴한다.

설명

isDown() 메소드는 keycode에 해당하는 키가 눌러져 있는지 확인하는 데 쓰인다. 이 메소드를 이용하면 키보드의 상태를 바로 확인할 수 있으며, 계속해서 키보드를 통해 입력을 받아야 하는 경우 또는 여러 키를 동시에 누르는 것을 감지해야 하는 경우에 많이 쓰인다.

isDown() 메소드에는 getCode()나 getAscii()와는 달리 동시에 여러 개의 키를 누르더라도 사용할 수 있다는 장점이 있다. 예를 들어 게임에서 Key.UP과 Key.RIGHT를 동시에 확인하여 우주선이 대각선 방향으로 움직이도록 만들 수도 있다. 테스트할 키의 배치에 따라 동시에 감지할 수 있는 키 개수의 최대값은 세 개 까지 줄어들 수도 있다.

예제

isDown() 메소드는 보통 프레임이 바뀌더라도 계속해서 업데이트되는 시스템을 만들 때 사용한다. 아래 코드에서는 화살표 키를 누르면 어떤 프레임에서도 우주선을 회전시키고 속도를 높이는 기능을 구현한다. 두 개의 키를 동시에 감지하려면 별도의 if 문을 사용해야 한다는 점에 주의하자. 이 예제에서는 왼쪽과 오른쪽 화살표 키를 동시에 누르면 오른쪽 화살표키는 무시한다. 하지만 위쪽 화살표 키는 별도의 if 선언문으로 확인하기 때문에 왼쪽이나 오른쪽 화살표 키와는 별도로 처리된다. 이 예제를 실제 프로그램에 적용한 것은 온라인 코드 창고에 올려 놓았다.

```
// 우주선에 사용할 코드
onClipEvent (enterFrame) {
    if (Key.isDown(Key.LEFT)) {           // 왼쪽 화살표
        _rotation -= 10;
    } else if (Key.isDown(Key.RIGHT)) {   // 오른쪽 화살표
        _rotation += 10;
    }
    if (Key.isDown(Key.UP)) {             // 위쪽 화살표
        thrust += 10;
    }
}
```

참조

Key.getCode(); 10장의 'keyDown'

Key.isToggled() 메소드Caps Lock, Num Lock, Scroll Lock이
활성화되어 있는지 확인하는 메소드

버전	플래시 5
문법	<code>Key.isToggled(keycode)</code>
인자	
<i>keycode</i>	정수 키코드. 보통 Caps Lock 키(20), Num Lock 키(144) 또는 Scroll Lock 키(145)의 키코드를 인자로 사용한다. 숫자 대신 <code>Key.CapsLock</code> 과 같은 상수를 사용해도 된다.

리턴 값

`keycode`에 해당하는 키가 활성화되어 있으면 `true`를, 그렇지 않으면 `false`를 리턴한다.

설명

`isToggled()` 메소드는 Caps Lock, Num Lock, Scroll Lock 키가 활성화되어 있는지 감지하는 메소드이다. 다른 키와는 달리 이 세 개의 키에는 활성화 여부를 알 수 있는 'on' 상태와 'off' 상태가 있다. `isToggled()` 메소드의 리턴 값으로부터 키의 기능이 활성화되어 있는지 알 수 있다(`isToggled()` 메소드를 다른 키에 사용해도 되지만 톨 기능 없는 키에서는 리턴 값이 무의미하다. 다른 키의 상태를 감지할 때는 `isDown()`, `getCode()` 또는 `getAscii()` 메소드를 이용해야 한다).

_leveln 전역 속성

플레이어에서의 문서 레벨

버전	플래시 3 이후
문법	<code>_level0</code> <code>_level1</code> <code>_level2</code> <code>...</code> <code>_leveln</code>
액세스	읽기 전용

설명

플래시에서는 여러 개의 .swf 파일을 동시에 로딩하여 화면에 표시할 수 있다. 각 .swf 파일은 문서 레벨 스택에서 서로 다른 레벨에 들어간다(여러 개의 .swf 파일이 스테이지에서 겹치게 되면 레벨 번호가 더 높은 파일이 번호가 낮은 파일을 가린다). `_leveln` 속성에는 플레이어의 문서 레벨에 로딩된 .swf 파일의 메인 타임라인에 대한 레퍼런스가 저장된다. 각 문서 레벨은 `_level0`, `_level1`, `_level2`와 같이 번호가 붙은 속성으로 표현된다.

플래시 플레이어에 처음으로 로딩된 원본 문서는 언제나 `_level0`으로 간주한다.

예제

`_leveln` 레퍼런스는 일반적으로 문서 스택의 다른 레벨에 있는 무비를 제어하는데 쓰인다. 예를 들어 다음과 같은 코드를 이용하면 2번 레벨에 있는 무비를 재생할 수 있다.

```
_level2.play();
```

다음과 같이 `_leveln`과 무비 클립 레퍼런스를 조합해서 사용하면 문서 스택의 어느 레벨에 있는 무비에 포함된 클립이라도 제어할 수 있다.

```
_level1.orderForm.titleBar.play();
```

또한 `_leveln` 속성을 `loadMovie()`나 `unloadMovie()`, `loadVariables()`, `print()`와 같은 함수에서 `target` 인자로 사용할 수도 있다. 레벨이 아직 만들어지지 않은 경우에는 레벨 레퍼런스를 괄호로 감싸야 한다. 그렇게 하지 않으면 존재하지 않는 레벨은 `undefined`로 간주되기 때문에 명령어가 현재 타임라인에 적용된다. 예를 들어 `_level0`의 메인 타임라인에서 다음과 같은 코드를 실행하면 `_level1`이 아직 만들어지지 않은 상태에서는 `_level0`에 다른 무비를 집어넣게 된다.

```
loadMovie("myMovie.swf", _level1);
```

따라서 레벨이 만들어져 있는지 확실하지 않은 경우에는 다음과 같이 하는 것이 안전하다.

```
loadMovie("myMovie.swf", "_level1"); // _level1이 없더라도 제대로 동작한다.
```

물론 `_level0`을 이용하여 다른 레벨에서 원래 레벨을 참조할 수도 있다.

```
startDrag(_level0, true);
```

참조

`loadMovie()`, `unloadMovie()`, `_root`; 13장의 ‘외부 무비 가져오기’, ‘무비와 인스턴스 스택 순서’

loadMovie() 전역 함수

외부 .swf 파일을 플레이어로 불러오는 함수

버전 플래시 4 이후. 플래시 5의 `loadMovie()` 함수는 플래시 4에서 대상 경로를 설정하고 Load Movie 액션을 사용하는 것과 같다.

문법

```
loadMovie(URL, target)
loadMovie(URL, target, method)
```

인자

URL 로딩할 외부 .swf 파일의 절대 경로 또는 상대 경로를 나타내는 문자열

target 외부 .swf 파일이 들어갈 문서 레벨이나 무비 클립을 나타내는 문자열. 이미 존재하는 무비 클립이나 문서 레벨에 대한 레퍼런스를 사용해도 된다(레퍼런스를 문자열 자리에 집어넣으면 경로로 자동 변환된다).

method 외부 스크립트로 변수를 보내는 방법을 나타내는 문자열. method에 쓸 수 있는 값에는 “GET”과 “POST”가 있다. 이 인자에는 변수나 다른 표현식은 사용할 수 없으므로 반드시 문자열 리터럴을 이용해야 한다. 독립형 플래시 플레이어에서는 method에서 지정한 방법에 상관없이 언제나 “GET” 메소드만을 사용한다(옵션).

설명

`loadMovie()` 함수는 URL 위치에 있는 외부 .swf 파일을 플래시 플레이어로 불러오는 함수이다.

target이 이미 있는 무비 클립에 대한 레퍼런스이거나 무비 클립을 가리키는 경로를 나타내는 문자열이면 .swf 파일이 그 클립으로 들어간다(원래 있던 내용은 지워진다). 무비를 현재 무비 클립으로 가져올 때는 target 인자에 비어있는 문자열을 사용하면 된다.

```
loadMovie("myMovie.swf", " ")
```

target이 이미 있는 문서 레벨(예: _level2)에 대한 레퍼런스 또는 문서 레벨의 경로를 나타내는 문자열(예: "_level2")이면, 주어진 문서 레벨로 .swf 파일을 불러들인다. 무비를 _level0으로 불러오면 플레이어에 들어있는 모든 내용이 지워지고 새로운 .swf 파일이 _level0으로 들어간다.

loadMovie()를 호출할 때 변수를 전송할 수도 있는데, 그러한 경우에는 보통 URL 자리에 주어진 변수를 바탕으로 .swf 파일을 리턴하는 스크립트가 들어간다. loadMovie()를 호출할 때 변수를 보내려면 method 인자를 포함시켜야 한다("GET" 또는 "POST"). "GET"을 이용하면 현재 무비 클립의 타임라인 변수를 스크립트 URL 뒤에 질의 문자열 형태로 덧붙여서 보낸다. "POST"를 이용하면 현재 무비 클립의 타임라인 변수가 HTTP POST 요청 헤더 뒤에 덧붙여서 전송된다. 하지만 독립형 플래시 플레이어에서는 "POST" 메소드를 사용할 수 없다. 대부분의 웹 서버에서는 URL의 길이를 255-1024자 정도로 제한하기 때문에, 큰 용량의 데이터를 전송할 때는 "GET" 대신 "POST"를 사용해야 한다.

"GET" 메소드를 사용하여 loadMovie() 함수를 호출할 때는 스크립트를 사용하지 않고도 웹 서버를 통해 새로 가져온 무비에 변수를 전달할 수 있다. 아래 예에서는 외부 무비 파일인 myMovie.swf를 플레이어 문서 스택의 1번 레벨에 로딩하고 현재 타임라인에 있는 변수들을 전달한다.

```
loadMovie("myMovie.swf", "_level1", "GET");
```

새로 로딩한 무비에 전달된 변수는 그 무비의 타임라인에서 정의된 변수처럼 사용할 수 있다. 하지만 이러한 방법은 loadMovie() 요청을 웹 서버에서 처리하는 경우에만 사용할 수 있다. 로컬 파일을 부를 때 loadMovie()에 "GET" 메소드를 인자로 전달하면 "Error opening URL"이라는 오류가 생긴다.

주의 사항

loadMovie()의 target 인자로 무비 클립이나 레벨에 대한 레퍼런스를 사용할 때 주의할 점이 있다. loadMovie()에 전달한 target 인자의 값이 undefined이면 loadMovie() 함수에서는 현재 타임라인에 .swf 파일을 불러오게 된다. target 레퍼런스가 비어있는 문자열인 경우에도 마찬가지로 현재 타임라인으로 .swf 파일을 불러온다. 특히 비어있는 새로운 레벨에 무비를 가져올 때 이러한 일이 자주 일어난다. 다음 코드를 살펴보자.

```
loadMovie("myMovie.swf", _level1);
```

이 선언문이 실행될 때 _level1 객체가 없다면 myMovie.swf는 _level1이 아닌 loadMovie() 선언문이 들어있는 타임라인으로 로딩된다. 이러한 문제가 일어나지 않도록 하려면 loadMovieNum()을 사용하면 된다. 또는 loadMovie()의 target 매개변수에 문자열을 전달해도 된다.

```
loadMovie("myMovie.swf", "_level1");
```

이렇게 하면 레벨이 없을 경우에는 자동으로 새로운 레벨이 만들어진다(모든 무비를 처음 재생할 때는 기본적으로 _level0만 있다). 자세한 내용은 13장의 '메소드와 전역 함수가 겹치는 문제'를 참조하기 바란다.

예제

```
loadMovie("myMovie.swf", "_level1"); // Place myMovie.swf on level 1
loadMovie("myMovie.swf", "_level0"); // Place myMovie.swf on level 0
loadMovie("myMovie.swf", "myClip"); // Place myMovie.swf into myClip
// 절대 경로를 이용하여 플레이어에 로딩된 내용을
// coolmovie.swf로 바꾼다.
loadMovie("http://www.yourflashsite.com/coolmovie.swf", "_level0");
// 윈도우 바탕화면에 있는 .swf 파일을 1번 레벨로 불러온다.
// file:/// 프로토콜을 사용할 때는 슬래시(/)를 사용한다는 점에 주의하자.
loadMovie("file:///C:/WINDOWS/Desktop/animation.swf", "_level1");
```

참조

loadMovieNum(), MovieClip.loadMovie(), unloadMovie(); 13장의 '외부 무비 가져오기'

loadMovieNum() 전역 함수

외부 .swf 파일을 특정 문서 레벨로 불러오는 함수

버전 플래시 3(플래시 4부터 method 매개변수가 포함됨. 플래시 5에서도 사용할 수 있다. loadMovieNum() 함수는 레벨 번호만을 받아들이는 플래시 3의 Load Movie에 해당한다).

문법

```
loadMovieNum(URL, level)
loadMovieNum(URL, level, method)
```

인자

URL 불러올 외부 .swf 파일의 절대 경로 또는 상대 경로를 나타내는 문자열

level .swf 파일을 집어넣을 문서 레벨을 나타내는 음이 아닌 정수 또는 정수 표현식

method 외부 스크립트에 변수를 보내는 방법을 나타내는 문자열. method에는 “GET”과 “POST”만 사용할 수 있다. 이 매개변수에는 변수나 표현식은 사용할 수 없고 문자열 리터럴만 전달해야 한다. 독립형 플래시 플레이어에서는 method로 지정한 방법과는 상관없이 “GET” 메소드만을 사용할 수 있다(옵션).

설명

loadMovieNum() 함수는 level을 숫자로 지정해야 한다는 점을 제외하면 loadMovie()와 거의 똑같다. 따라서 loadMovieNum() 함수에서는 문서 레벨에만 무비를 불러올 수 있을 뿐 호스트 클립에는 문서를 불러올 수 없다. 지정된 레벨이 존재하지 않는다면 자동으로 새로운 레벨을 만든다. 주어진 레벨이 존재하는 경우에는 그 자리에 새로운 .swf 파일이 들어간다. _level1이 없어도 _level2에 무비를 로딩할 수 있다.

loadMovieNum() 함수는 로딩된 무비의 레벨을 동적으로 설정할 때도 사용할 수 있다.

```
var x = 3;
loadMovieNum("myMovie.swf", x);
```


loadMovie() 함수를 이용하는 경우에도 다음과 같이 문자열 합치기 연산을 활용하면 위와 똑같은 작업을 처리할 수 있다.

```
loadMovie("myMovie.swf", "_level" + x);
```

참조

loadMovie(), MovieClip.loadMovie()

loadVariables() 전역 함수

외부의 변수를 가져오는 함수

버전 플래시 4 이후

문법 `loadVariables(URL, target)`
`loadVariables(URL, target, method)`

인자

URL 변수를 가져오기 위한 경로(변수를 리턴하는 서버 스크립트 또는 변수가 저장된 텍스트 파일)를 나타내는 문자열

target 로딩한 변수를 정의할 무비 클립이나 문서 레벨의 경로를 나타내는 문자열. 무비 클립이나 문서 레벨에 대한 레퍼런스를 사용해도 된다(레퍼런스를 문자열이 들어갈 자리에 넣으면 경로로 자동 변환된다).

method 변수를 외부 스크립트로 보낼 때 사용할 메소드를 나타내는 문자열. 이 값을 지정해주면 현재 타임라인의 변수가 스크립트로 전송되고 그 스크립트에서 보내온 변수는 target에 저장된다. 이 값을 생략하면 변수를 받기만 하고 보내지는 않는다. method에는 "GET" 또는 "POST"를 변수나 표현식이 아닌 문자열 리터럴로 전달해야 한다. 독립형 플래시 플레이어에서는 method에 지정한 방법과는 상관없이 "GET" 메소드만 사용한다(옵션).

설명

보통 변수는 액션스크립트를 이용하여 무비 안에서 정의한다. 하지만 loadVariables()를 이용하면 텍스트 파일이나 펄 스크립트와 같은 서버측 애플리케이션

으로부터 무비로 변수를 불러올 수 있다. loadVariables()를 이용하여 가져온 변수는 target으로 지정한 무비 클립이나 레벨의 영역에 속하게 되며, 모든 변수가 문자열 데이터형으로 간주된다. 외부에서 가져온 변수들을 현재 타임라인에 추가하려면 target 인자에 비어있는 문자열을 사용하면 된다.

```
loadVariables("myVars.txt", ""); // myVars.txt로부터 현재 타임라인으로
// 변수를 가져온다.
```

변수를 텍스트 파일에서 가져오는 스크립트에서 가져오는 상관없이 변수의 형식은 다음과 같은 URL 인코딩 규칙을 만족시켜야 한다.

- 모든 변수 이름과 값은 등호(=)로 분리하며 그 사이에는 공백이 없어야 한다(예: firstName=stephen).
- 여러 개의 변수 이름/값 쌍은 & 기호로 분리한다(예: firstName=stephen&lastName=burke).
- 스페이스는 모두 덧셈(+) 기호로 표시한다.
- 스페이스, 숫자(0-9), Latin 1 글자(a-z, A-Z)를 제외한 모든 문자는 %xx 형태의 16진수 이스케이프 시퀀스로 표기해야 한다(xx는 그 문자의 Latin 1 코드 포인트).

예를 들어 loadVariables()를 이용하여 플래시로 불러올 텍스트 파일을 만들 때는 다음과 같은 형식으로 저장해야 한다. 아래 예에서 가져오는 변수는 name과 address이며 변수 값은 각각 “stephen”과 “65 nowhere st!”이 된다.

```
name=stephen&address=65+nowhere+st%21
```

loadVariables()에서 사용할 텍스트 파일은 위에 나온 것과 같은 URL 인코딩된 변수를 포함하는 일반적인 텍스트 파일이다. 외부 텍스트 파일에서 변수를 가져올 때는 loadVariables() 함수를 호출할 때 그 파일의 경로를 URL 인자로 지정하면 된다. 예를 들면 다음과 같다.

```
// myVariables.txt에서 메인 무비 타임라인으로 변수를 불러온다.
loadVariables("myVariables.txt", "_root");
```

loadVariables() 함수를 URL 인코딩된 변수를 출력하는 스크립트나 서버 애플리케이션과 함께 사용할 수도 있다. loadVariables() 함수를 호출한 결과로 스크립트에서 플래시 무비에 데이터를 보낼 때는 데이터의 MIME 유형을 “application/x-www-urlform-encoded”로 설정해야 한다. 펄 스크립트를 이용한다면 다음과 같은 식으로 MIME 유형을 설정하면 된다.

```
print "Content-type: application/x-www-urlform-encoded\n\n";
```

loadVariables()라는 이름만 보면 가져오는 것만 할 수 있을 것 같지만, 이 함수를 이용하여 서버측 스크립트에 변수를 보낼 수도 있다. 현재 타임라인에서 정의한 모든 변수를 스크립트로 보낼 때는 loadVariables() 함수를 호출할 때 method 인자를 “GET” 또는 “POST”로 설정하면 된다. 변수는 URL 인코딩된 형식으로 전송된다. method를 “GET”으로 설정하면 변수를 URL 뒤에 붙는 질의 문자열 형태로 전달한다. method를 “POST”로 설정하면 변수를 HTTP POST 요청 헤더 뒤에 덧붙여서 전달한다. 독립형 플래시 플레이어에서는 “POST” 메소드를 사용할 수 없다. 대부분의 웹 서버에서는 URL의 길이를 255-1024 글자 이내로 제한하기 때문에, 큰 데이터를 보낼 때는 “POST” 메소드를 사용하는 것이 좋다.

보안상의 이유로 loadVariables() 함수는 무비를 다운로드한 도메인에 있는 호스트에만 사용할 수 있다. loadVariables()를 사용할 수 있는 호스트와 관련된 규칙은 [표 R-8]에 나와 있다. 하지만 이러한 제약 조건은 플래시 플레이어 브라우저 플러그인과 ActiveX 컨트롤에 대해서만 적용된다. 독립형 플래시 플레이어를 사용하는 경우에는 데이터를 주고받을 수 있는 도메인에 대한 제한이 없다.

[표 R-8] 도메인을 기반으로 한 loadVariables()의 보안 관련 제약 조건

무비를 받은 도메인	연결할 호스트	연결 가능 여부
www.somewhere.com	www.somewhere.com	가능
www.somewhere.com	other.somewhere.com	가능
www.somewhere.com	www.somewhere-else.com	불가능
www.somewhere.com	somewhere.com	가능
somewhere.com	www.somewhere.com	가능

도메인 제한은 플래시 보안을 강화하기 위해 고안된 방법이다. 하지만 플래시와 Y라는 사이트 사이를 연결하는 프록시 스크립트를 X라는 사이트에서 실행시키거나

X 사이트에서 Y 사이트로 연결되는 DNS 앨리어스를 이용하면 이러한 제약 조건에서 빠져나갈 수도 있다. 이와 관련된 자세한 내용은 아래 URL에서 찾을 수 있다.

http://www.macromedia.com/support/flash/ts/documents/loadvars_security.htm

주의 사항

loadVariables()를 여러 번 호출하여 같은 스크립트 URL을 반복적으로 불러오면 브라우저의 캐시 기능 때문에 서버에 있는 새로운 데이터를 가져오지 못하게 되는 경우도 있다. 이러한 문제를 방지하려면 매번 loadVariables() 함수를 호출할 때마다 별 의미 없는 변수를 추가하여 각 URL이 서로 다른 형태가 되도록 만들면 된다. 예를 들어 다음과 같이 밀리초 단위의 시간을 덧붙여 똑같은 URL이 생기는 것을 방지할 수 있다.

```
loadVariables("http://www.mysite.com/cgi-bin/myScript.pl?cacheKiller="
+ getTimer(), serverResponse);
```

버그

맥킨토시용 인터넷 익스플로러 4.5에서는 POST 메소드를 사용할 수 없다. 맥킨토시용 인터넷 익스플로러 5.0에서는 이러한 문제가 해결되었다.

참조

loadVariablesNum(), MovieClip.loadVariables(); '17장. 플래시 폼'

loadVariablesNum() 전역 함수

외부 변수를 특정 문서 레벨에 추가하는 함수

버전 플래시 5(플래시 4에서는 Load Variables 액션을 이용하여 특정 문서 레벨에 변수를 추가할 수 있다)

문법 loadVariablesNum(URL, level)
loadVariablesNum(URL, level, method)

인자

URL 변수를 가져오기 위한 경로(변수를 리턴하는 서버 스크립트 또는 변수가 저장된 텍스트 파일)를 나타내는 문자열

level	불러온 변수를 정의할 문서 레벨을 나타내는 음이 아닌 정수 또는 정수 표현식
method	변수를 외부 스크립트로 보낼 때 사용할 메소드를 나타내는 문자열. 이 값을 지정해주면 현재 타임라인의 변수가 스크립트로 전송되고, 그 스크립트에서 보내온 변수가 level에 저장된다. 이 값을 생략하면 변수를 받기만 하고 보내지는 않는다. method에는 “GET” 또는 “POST”를 변수나 표현식이 아닌 문자열 리터럴로 전달해야 한다. 독립형 플래시 플레이어에서는 method에 지정한 방법과는 상관없이 “GET” 메소드만 사용한다(옵션).

설명

loadVariablesNum() 함수는 변수를 정의할 레벨을 문자열이 아닌 숫자로 지정한다는 점을 제외하면 loadVariables() 함수와 거의 똑같다. 즉 loadVariablesNum() 함수에서는 문서 레벨에만 변수를 추가할 수 있고 무비 클립에는 변수를 추가할 수 없다. level 인자는 다음과 같이 동적으로 지정할 수 있다.

```
var myLevel = 2;
loadVariablesNum("myVars.txt", myLevel);
```

loadVariables() 함수를 이용하는 경우에도 문자열을 합치면 비슷한 방식으로 작업을 처리할 수 있다.

```
loadVariables("myVars.txt", "_level" + myLevel);
```

참조

loadVariables()

Math 객체

수학 함수 및 상수 액세스

버전	플래시 5
문법	Math. <i>propertyName</i> Math. <i>methodName</i> ()

속성

<i>E</i>	자연로그의 밑인 e 값. 대략 2.71828 정도
<i>LN10</i>	10의 자연로그 값(log _e 10). 대략 2.30259 정도
<i>LN2</i>	2의 자연로그 값(log _e 2). 대략 0.69315 정도
<i>LOG10E</i>	10을 밑으로 하는 e의 로그 값. 대략 0.43429 정도
<i>LOG2E</i>	2를 밑으로 하는 e의 로그 값. 대략 1.44270. 정도 Math.LOG2E의 버그 부분 참조
<i>PI</i>	원의 둘레와 지름 사이의 비율. 대략 3.14159 정도
<i>SQRT1_2</i>	루트 2의 역수. 대략 0.70711 정도
<i>SQRT2</i>	루트 2. 대략 1.41421 정도

메소드

<i>abs()</i>	어떤 숫자의 절대값을 리턴하는 메소드
<i>acos()</i>	어떤 숫자의 아크코사인을 리턴하는 메소드
<i>asin()</i>	어떤 숫자의 아크사인을 리턴하는 메소드
<i>atan()</i>	어떤 숫자의 아크탄젠트를 리턴하는 메소드
<i>atan2()</i>	어떤 점과 x축이 이루는 각도를 리턴하는 메소드
<i>ceil()</i>	어떤 숫자 이상의 가장 작은 정수를 리턴하는 메소드(올림)
<i>cos()</i>	어떤 각도의 코사인 값을 구하는 메소드
<i>exp()</i>	e의 멱수를 구하는 메소드
<i>floor()</i>	입력된 값 이하의 최대 정수를 리턴하는 메소드(내림)
<i>log()</i>	주어진 숫자의 자연로그 값을 리턴하는 메소드
<i>max()</i>	두 숫자 중 더 큰 값을 리턴하는 메소드
<i>min()</i>	두 숫자 중 더 작은 값을 리턴하는 메소드
<i>pow()</i>	주어진 숫자의 멱수를 구하는 메소드
<i>random()</i>	0과 1 사이에서 무작위로 선택한 부동소수점 소수를 리턴하는 메소드
<i>round()</i>	주어진 숫자에 가장 가까운 정수를 구하는 메소드(반올림)

<code>sin()</code>	주어진 각도의 사인 값을 구하는 메소드
<code>sqrt()</code>	주어진 숫자의 제곱근을 구하는 메소드
<code>tan()</code>	주어진 각도의 탄젠트 값을 구하는 메소드

설명

Math 객체를 이용하면 액션스크립트에 내장된 수학 함수(메소드를 통해 사용)와 상수 값(속성을 통해 사용)을 이용할 수 있다. 이러한 함수와 상수는 복잡한 계산을 간단하게 처리하기 위해 쓰인다.

Math 객체의 속성과 메소드는 무비를 플래시 4 형식으로 저장하는 경우에도 사용할 수 있으며 그러한 경우에는 플래시에서 자동으로 적절히 계산을 할 수 있도록 만들어준다(원래 플래시 4에는 그러한 기능이 없다). 플래시 4에서 이렇게 계산한 값은 거의 정확하긴 하지만 플래시 5에서 계산한 결과와 약간 다를 수도 있다. 플래시 4에서 계산한 값은 그래픽 디스플레이 정도의 용도로는 충분히 사용할 수 있지만 금융이나 공학용 계산에 사용하기에는 약간 부족하다.

삼각함수에서는 각도를 라디안으로 사용해야 하지만 플래시의 MovieClip._rotation 속성은 도 단위로 저장된다. 한 바퀴(360도)는 2π 이다(1 라디안은 대략 57.3도 정도임). 다음과 같은 공식을 이용하면 라디안을 도로 변환할 수 있다.

```
degrees = (radians / Math.PI) * 180;
```

반대로 도를 라디안으로 변환할 때는 다음 공식을 이용하면 된다.

```
radians = (degrees / 180) * Math.PI;
```

참조

Math.atan2(), Math.cos(); 4장의 '숫자 유형'

Math.abs() 메소드

주어진 숫자의 절대값을 계산하는 메소드

버전	플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)
문법	Math.abs(x)

인자

x 양 또는 음의 숫자

리턴 값

x 의 절대값(크기가 x 인 양수)

설명

`abs()` 메소드에서는 x 와 0 사이의 거리(절대값)를 계산한다. 양수는 그대로 리턴하고 음수는 같은 크기의 양수로 변환하여 리턴한다. 주로 부호에 상관없이 두 값의 차를 구할 때 많이 사용한다. 예를 들면 거리는 항상 양수로 생각하기 때문에, 두 점 사이의 거리를 구할 때 이 메소드를 이용하기도 한다.

예제

```
Math.abs(-5); // Returns 5

// 두 숫자 사이의 차를 구한다.
function diff (num1, num2) {
    return Math.abs(num1-num2);
}

diff(-5, 5); // 10이 리턴된다.
```

Math.acos() 메소드

주어진 값의 아코사인 값을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.acos(x)`

인자

x -1.0과 1.0 사이의 값(어떤 각도의 코사인은 -1.0과 1.0 사이이므로)

리턴 값

코사인 값이 x 인 각도(라디안 단위). x 가 -1.0과 1.0 사이에 속하지 않으면 NaN을 리턴한다.

설명

아크코사인 함수(\cos^{-1} 이라고도 씀)는 코사인 함수의 역함수이다. 주어진 코사인 값을 가지는 각도를 라디안 단위로 리턴한다. 리턴 값의 범위는 0과 π 사이(즉, 0과 3.14159... 사이)이다.

예제

```
trace (Math.acos(1.0)); // 0이 출력된다.  
trace (Math.acos(0.0)); // 1.5707... (즉, pi/2)이 출력된다.
```

참조

Math.asin(), Math.atan(), Math.cos()

Math.asin() 메소드

주어진 값의 아크사인 값을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.asin(*x*)

인자

x -1.0과 1.0 사이의 값(어떤 각도의 사인 값은 -1.0과 1.0 사이이므로)

리턴 값

사인 값이 *x*인 각도(라디안 단위). *x*가 -1.0과 1.0 사이에 속하지 않으면 NaN을 리턴한다.

설명

아크사인 함수(\sin^{-1} 라고도 씀)는 사인 함수의 역함수이다. 주어진 사인 값을 가지는 각도를 라디안 단위로 리턴한다. 리턴 값의 범위는 $-\pi/2$ 와 $\pi/2$ 사이이다.

예제

```
trace (Math.asin(1.0)); // 1.5707... (즉, pi/2)이 출력된다.  
trace (Math.asin(0.0)); // 0이 출력된다.
```

참조

Math.acos(), Math.atan(), Math.sin()

Math.atan() 메소드

주어진 값의 아크탄젠트 값을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.atan(x)**인자****x** -Infinity와 Infinity 사이의 값(어떤 각도의 탄젠트 값은 -Infinity와 Infinity 사이이므로). -Infinity, +Infinity도 x 값으로 사용할 수 있다.**리턴 값**

탄젠트 값이 x인 각도를 라디안 단위로 리턴한다.

설명아크탄젠트 함수(tan-1라고도 씀)는 탄젠트 함수의 역함수이다. 주어진 값이 탄젠트 값인 각도를 라디안 단위로 리턴한다. 리턴 값의 범위는 $-\pi/2$ 와 $\pi/2$ 사이이다.**예제**

```

trace (Math.atan(1.0));           // 0.78539...가 출력된다.
trace (Math.atan(0.0));           // 0이 출력된다.
trace (Math.atan(-Infinity));     // -1.5707... (즉, -pi/2)이 출력된다.

```

참조

Math.acos(), Math.asin(), Math.tan()

Math.atan2() 메소드

점을 기준으로 각도를 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.atan2(y, x)

인자

y	어떤 점의 y 좌표값
x	어떤 점의 x 좌표값

리턴 값

(x , y) 점과 원의 중심(원점)을 연결한 직선과 x 축 양의 방향이 이루는 각도를 반시계 방향으로 측정한 각도를 라디안 단위로 리턴한다. $-\pi$ 부터 π 사이의 값이 리턴된다(음수는 x 축 아래쪽을 의미한다).

설명

`atan2()` 메소드는 `atan()`과 마찬가지로 아크탄젠트 값을 구하는 메소드이지만, 좌표평면에 있는 어떤 점의 x 와 y 좌표값을 인수로 사용한다. 즉 다음과 같은 식으로 아크탄젠트 값을 계산하는 것은

```
Math.atan2(9, 3); // 1.24904577239825
```

다음과 같이 9/3(즉 3)을 인자로 사용하여 `atan()` 메소드를 호출하는 것과 같다.

```
Math.atan(3); // 똑같은 값
```

주의 사항

`atan2()`에서는 첫 번째 인자가 y 좌표값이고 두 번째 인자가 x 좌표값이라는 점에 주의하자. 순서가 이렇게 되어있는 이유는 탄젠트가 y 를 x 로 나눈 값이기 때문이다.

예제

무비 클립이 어떤 움직이는 대상을 향하도록 만들 때 `atan2()` 메소드를 사용할 수 있다. 아래 예제(온라인 코드 창고에서 구할 수 있음)에서 현재 클립을 마우스 포인터 방향으로 회전시키는 방법을 알 수 있다. 게임에서 이러한 방법을 이용하여 적 우주선이 주인공 우주선을 향해 회전하도록 만들 수도 있다.

```
// 무비 클립을 마우스를 향해 회전시키는 코드
onClipEvent (load) {
    // 라디안을 도로 변환한다. 2*pi는 360도이다.
    function radiansToDegrees(radians) {
        return (radians/Math.PI) * 180;
    }
}
```

```

    }

    onClipEvent (enterFrame) {
        // 부모 클립의 등록지점에 대해 상대적인 x, y 값을
        // 저장하기 위한 point 객체를 만든다.
        point = {x:_x, y:_y};
        // 지역(부모 클립 기준) 좌표를 전역(스테이지 기준) 좌표로 변환
        _parent.localToGlobal(point);
        // 클립이 등록된 지점과 마우스와의 거리를 계산한다.
        // point of this clip and the mouse
        deltaX = _root._xmouse - point.x;
        deltaY = _root._ymouse - point.y;
        // 클립이 등록된 지점과 마우스 위치를 연결한 직선의
        // 각도를 계산한다.
        rotationRadian = Math.atan2(deltaY, deltaX);
        // 라디안으로 구한 각도를 도 단위로 변환한다.
        rotationAngle = radiansToDegrees(rotationRadian); // 위 함수 참조
        // 클립을 회전시켜서 마우스를 가리키도록 만든다.
        this._rotation = rotationAngle;
    }

```

Math.ceil() 메소드

주어진 숫자 이상의 가장 작은 정수를 구하는 메소드 (올림)

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.ceil(x)

인자

x 숫자

리턴 값

x 이상의 가장 작은 정수

설명

ceil()(올림) 메소드에서는 부동소수점 소수를 x 이상의 가장 작은 정수로 변환한다.

예제

```
Math.ceil(1.00001); // 2를 리턴한다.  
Math.ceil(5.5);     // 6을 리턴한다.  
Math.ceil(-5.5);    // -5를 리턴한다.
```

참조

Math.floor(), Math.round()

Math.cos() 메소드

주어진 각도의 코사인 값을 계산하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.cos(theta)`

인자

theta 0 이상 2π 이하의 각도(라디안 단위)

리턴 값

theta의 코사인 값(-1.0 이상, 1.0 이하)을 리턴한다.

설명

cos() 함수는 주어진 각도의 코사인 값을 리턴한다. 직각 삼각형에서 어떤 각도의 코사인 값은 꼭지점에 붙어있는 변의 길이를 빗변의 길이로 나눈 값이다.

주의 사항

cos() 함수를 호출할 때는 도가 아닌 라디안 단위를 사용해야 한다.

예제

```
trace (Math.cos(0));           // 1.0이 출력된다.  
trace (Math.cos(Math.PI));    // -1.0이 출력된다.
```

cos() 함수를 sin() 함수와 함께 사용하면 클립을 원형으로 움직일 때 원 위 점의 좌표를 구할 수 있다. 원의 반지름이 r이고 각도(x축 양의 방향을 기준으로 반시계 방향으로 구한 값)가 θ 이면 그 점의 좌표는 $(r*\cos\theta, r*\sin\theta)$ 이다.

```
// 1번 프레임의 코드
var radius = 100;           // 원형 경로의 반지름
var centerX = 275;          // 원형 경로의 중심의 수평 위치
var centerY = 200;          // 원형 경로의 중심의 수직 위치
var rotAngleDeg = 0;         // 객체의 각도(도 단위) x축을 기준으로 반시계
                             // 방향으로 구한 값
var rotAngRad;              // rotAngleDeg을 라디안 단위로 계산한 값

// 도 단위를 라디안 단위로 변환한다. 360도는 2*pi이다.
function degreesToRadians(degrees) {
    return (degrees/180) * Math.PI;
}

// 2번 프레임의 코드
// 회전 각도를 5도 증가시킨다.
rotAngleDeg += 5;

// 객체의 위치를 변경시킨다. 플래시에서는 Y축의 방향이 아래를 향하므로
// 새로운 위치를 구할 때 y 값을 감소시켜야 한다.
rotAngRad= degreesToRadians(rotAngleDeg);
ball._x = centerX + Math.cos(rotAngRad) * radius;
ball._y = centerY - Math.sin(rotAngRad) * radius;

// 3번 프레임의 코드
// 2번 프레임으로 돌아가서 객체를 다시 움직인다.
gotoAndPlay(2);
```

참조

Math.acos(), Math.sin(), Math.tan()

Math.E 속성

상수 e(자연로그의 밑)

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.E

설명

E 속성에는 자연로그 밑 근사값(대략 2.71828 정도)이 들어있다. 보통 수학에서는 이 값을 e로 표시한다. π 와 마찬가지로 무리수이며 어떤 값의 증감과 연관된 수

학식에서 많이 쓰인다. 부동소수점 소수를 지수 형태로 표현할 때 쓰이는 E와 혼동하지 않도록 주의하자(4장의 '부동소수점 리터럴' 참조). 이 두 가지는 전혀 상관이 없다.

참조

`Math.log()`, `Math.LN10()`, `Math.LN2()`

Math.exp() 메소드

e의 멱승을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.exp(x)`

인자

x `Math.E`의 멱승을 구할 승수

리턴 값

`Math.E`를 *x*번 곱한 값(*e*의 *x*승)

Math.floor() 메소드

어떤 수 이하의 최대 정수를 구하는 메소드(내림)

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.floor(x)`

인자

x 숫자

리턴 값

x 이하의 최대 정수

설명

`floor()` 메소드는 부동소수점 소수를 그 이하의 최대 정수로 변환(내림)하는 메소드이다.

예제

```
Math.floor(1.99999); // 1
Math.floor(5.5);     // 5
Math.floor(-5.5);    // -6

function minutesToHMM (minutes) {
    var hours = Math.floor(minutes/60);
    minutes -= hours * 60;
    minutes = minutes < 10 ? "0" + minutes : minutes;
    return hours + ":" + minutes;
}
```

참조

Math.ceil(), Math.round()

Math.LN10 속성

10의 자연로그 값(loge10). 대략 2.30259 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.LN10

설명

LN10 속성은 10의 자연로그 값(e를 밑으로 하는 10의 로그값)을 나타낸다. 그 값은 대략 2.30259 정도이다.

Math.LN2 속성

2의 자연로그 값(loge2). 대략 0.69315 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.LN2

설명

LN2 속성은 2의 자연로그 값(e를 밑으로 하는 2의 로그값)을 나타낸다. 그 값은 대략 0.69315 정도이다.

Math.log() 메소드

주어진 숫자의 자연 로그를 계산하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** `Math.log(x)`**인자*****x*** 양수**리턴 값***x*의 자연로그 값**설명**

`log()` 메소드에서는 주어진 숫자의 자연로그 값(*e*를 밑으로 하는 로그값)을 계산한다. 다음과 같이 하면 10을 밑으로 하는 로그를 구할 수 있다.

예제

```
trace (Math.log(Math.E));    // 1

// 주어진 숫자의 상용로그(10을 밑으로 하는 로그)를 구한다.
function log10 (x) {
    return (Math.log(x) / Math.log(10));
}
```

Math.LOG10E 속성10을 밑으로 하는 *e*의 로그값. 대략 0.43429정도**버전** 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** `Math.LOG10E`**설명**

LOG10E는 *e*의 상용 로그(10을 밑으로 하는 *e*의 로그)를 나타내며 대략 0.43429 정도이다.

Math.LOG2E 속성

2를 밑으로 하는 e의 로그값. 대략 1.44270 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.LOG2E**설명**

LOG2E는 2를 밑으로 하는 2의 로그값(log2e)을 나타내며 대략 1.44270 정도이다.

버그

플래시 5r30에는 LOG2E에서 LN2(0.69315)를 리턴하는 버그가 있다.

Math.max() 메소드

두 수 가운데 더 큰 수를 리턴하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.max(x, y)**인자***x* 숫자*y* 숫자**리턴 값**

x와 y 중 더 큰 값

예제

```
Math.max(5, 1);    // 5
Math.max(-6, -5);  // -5
```

다음과 같은 코드를 이용하면 어떤 값의 범위를 제한할 수 있다.

```
function constrainToRange (checkVal, minVal, maxVal) {
    return Math.min(Math.max(checkVal, minVal), maxVal);
}
// 슬라이더를 스테이지 영역으로 제한한다.
mySlider._x = constrainToRange (mySlider._x, 0, 550);
```

아래 코드는 최대값을 리턴하는 함수이다.

```
function maxInArray (checkArray) {  
    maxVal = -Number.MAX_VALUE; // maxVal에 매우 작은 값을 대입하여  
                                // 초기화한다,  
    for (var i = 0; i < checkArray.length; i++) {  
        maxVal = Math.max(checkArray[i], maxVal);  
    }  
    return maxVal;  
}  
  
trace(maxInArray([2,3,66,4,342,-90,0])); // 342가 출력된다.
```

참조

Math.min()

Math.min() 메소드

두 수 가운데 더 작은 수를 리턴하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.min(x, y)

인자

x 숫자

y 숫자

리턴 값

x와 y 중 더 작은 값

예제

```
Math.min(5, 1); // 1  
Math.min(-6, -5); // -6
```

연습 문제: Math.max()에 나온 예제를 수정하여 배열에 저장된 값 중 최소값을 리턴하는 함수를 만들어 보자.

참조

Math.max()

Math.PI 속성

원의 둘레와 지름의 비율. 대략 3.14159 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.PI**설명**PI 속성은 원의 둘레와 지름의 비율인 원주율 π 를 나타낸다.**예제**

Math.PI는 주로 원의 넓이를 구할 때 쓰인다.

```
function circleArea (radius) {
    // PI에 반지름의 제곱을 곱한 값은
    // Math.PI * Math.pow(radius, 2)로 써도 된다.
    return Math.PI * (radius * radius);
}
```

Math.pow() 메소드

어떤 수의 멍수를 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** Math.pow(*base*, *exponent*)**인자****base** 밑을 나타내는 수**exponent** 지수를 나타내는 수**리턴 값**

base의 exponent승을 나타내는 숫자

설명

`pow()` 메소드를 이용하면 숫자의 임의의 승을 구할 수 있다. `exponent`가 음수인 경우에는 $1 / (\text{baseabs}(\text{exponent}))$ 를 리턴한다. `exponent`가 반드시 정수를 써야 하는 것은 아니므로 어떤 숫자의 제곱근을 구할 수도 있다(양의 제곱근만 리턴한다).

예제

```
Math.pow(5, 2);    // 25 (5의 제곱)
Math.pow(5, 3);    // 125 (5의 3승)
Math.pow(5, -2);   // 0.04 (1을 25로 나눈 값)
Math.pow(8, 1/3);  // 2 (8의 3승근)
Math.pow(9, 1/2);  // 3 (9의 제곱근)
Math.pow(10, 6);   // 1000000 (1e6으로 쓸 수도 있다)
```

버그

플래시 5r30에서는 `base`가 음수인 경우에 `Math.pow()`가 제대로 동작하지 않는다. 예를 들어 `Math.pow(-2, 2)`를 계산하면 원래 4가 나와야 하지만 NaN을 리턴한다.

참조

`Math.exp()`, `Math.sqrt()`, `Math.SQRT2`, `Math.SQRT1_2`

Math.random() 메소드

0에서 1.0까지 난수를 발생시키는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.random()`

리턴 값

0.0 이상, 1.0 미만의 부동소수점 소수

설명

`random()` 메소드를 이용하면 스크립트에서 여러 액션 가운데 하나를 임의로 선택할 수 있도록 난수를 만들 수 있다. `random()` 메소드는 0 이상, 0.99999... 이하의 값을 무작위로 발생시키며 이 값을 가지고 원하는 범위의 난수를 만들 수 있다. 예를 들어 0과 5 사이의 정수를 난수로 발생시킬 때는 다음과 같이 할 수 있다.

```
Math.floor(Math.random() * 6)
```

1과 6 사이의 정수를 난수로 발생시킬 때는 다음과 같이 하면 된다.

```
Math.floor(Math.random() * 6) + 1
```

아래 함수에서는 0과 1 사이의 부동소수점 소수 대신 주어진 영역에 속하는 정수를 난수로 발생시킨다.

```
// minVal 이상, maxVal 이하의 값을 리턴한다.
function myRandom (minVal, maxVal) {
    return minVal + Math.floor(Math.random() * (maxVal + 1 - minVal));
}

// 함수를 호출하는 방법
dieRoll = myRandom(1, 6); // 주사위 시뮬레이션
trace(dieRoll);

// 두 개의 주사위를 사용할 때는 다음과 같이 할 수 있다.
twoDice = myRandom(2, 12); // 최소값이 1이 아니라 2가 된다.

// 두 주사위의 값을 따로 리턴하려면 다음과 같이 한다.
function rollTwoDice () {
    return [myRandom(1, 6), myRandom(1, 6)];
}
```

플래시 5 플레이어의 빌드 30에 있는 버그 때문에 이러한 접근법은 큰 문제를 일으킬 수도 있다. 빌드 30에 있는 random() 메소드에서는 0.0 이상, 1.0 이하의 난수를 발생시킨다. 따라서 random()에서 리턴된 값에 정수 n을 곱하면 0.0 이상 n 이하의 값이 생긴다. 위 예제에서는 Math.random()에 6을 곱했으므로 난수의 범위가 0.0 이상, 6.0 이하가 된다. 이 값에 대해 floor() 메소드를 호출하면 0 이상 n 이하(위 예제에서는 0 이상 6 이하)의 정수가 생긴다. 그러면 난수의 분포가 부정확해진다. n이 나올 확률은 다른 정수들이 나올 확률보다 훨씬 낮기 때문이다.

아래에 나온 myRandom() 함수에서는 Math.random()에서 1.0을 리턴하면 그 값을 무시하는 방식으로 그 문제를 방지한다.

```
// minVal 이상, maxVal 이하의 정수를 난수로 발생한다.
function myRandom (minVal, maxVal) {
    do {
        r = Math.random(); // r이 1.0이 아닌 값이 될 때까지 난수를 발생시킨다.
    } while (r == 1);
}
```

```
    return minVal + Math.floor(r * (maxVal + 1 - minVal));  
}
```

// 위 함수를 호출한다.

```
dieRoll = myRandom(1, 6); // 플래시 5 빌드 30에서 사용해도 되는 함수  
// 주사위와 같은 효과를 낸다.
```

주의 사항

Math.random() 메소드는 이제는 더 이상 쓰이지 않는 플래시 4의 random 함수를 대신하는 메소드이다.

예제

Math.random()은 타임라인에서 플레이헤드를 무작위로 선택한 프레임으로 옮길 때 종종 쓰인다. 아래 코드에서는 위에서 만든 myRandom() 함수를 이용하여 플레이헤드를 무작위로 선택한 프레임으로 보낸다.

```
// 함수를 호출하여 10 이상 20 이하의 난수를 발생시킨다.
```

```
var destinationFrame = myRandom(10, 20);
```

```
// 플레이헤드를 선택된 프레임으로 보낸다.
```

```
gotoAndStop(destinationFrame);
```

Math.round() 메소드

주어진 숫자에 가장 가까운 정수를
계산하는 메소드(반올림)

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.round(x)

인자

x 숫자

리턴 값

수학적으로 *x*에 가장 가까운 정수(*x*가 정수인 경우에는 *x* 자신). *x*의 소수 부분이 정확하게 0.5이면(즉 *x*보다 큰 정수와 작은 정수의 정확하게 중간에 있을 때) *x*보다 큰 쪽의 정수를 리턴한다.

설명

`round()` 메소드는 일반적인 반올림을 처리하는 메소드로 부동소수점 소수를 가장 가까운 정수로 변환한다. 양수인 경우에는 소수 부분이 0.5보다 작을 때, 음수인 경우 0.5보다 클 때는 소수점 이하 부분을 버린다. 양수에서 소수 부분이 0.5 이상일 때, 음수에서 0.5 이하일 때는 소수점 이하 부분을 올린다.

예제

```
Math.round(1.4);    // 1
Math.round(1.5);    // 2
Math.round(1.6);    // 2
Math.round(-5.4);   // -5
Math.round(-5.5);   // -5
Math.round(-5.6);   // -6
```

참조

`int()`, `Math.ceil()`, `Math.floor()`

Math.sin() 메소드

주어진 각도의 사인 값을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.sin(theta)`

인자

theta 0 이상 2π 이하의 각도(도 단위가 아니라 라디안 단위임)

리턴 값

*theta*의 사인 값(결과의 범위는 -1.0 이상, 1.0 이하이다)

설명

`sin()` 메소드는 주어진 각도의 사인 값을 리턴하는 메소드이다. 직각 삼각형의 경우에는 주어진 각과 마주보는 변의 길이를 빗변의 길이로 나눈 값이 사인 값이 된다.

주의 사항

`sin()`을 사용할 때는 각도를 도 단위가 아닌 라디안 단위로 입력해야 한다는 점에 주의하자.

예제

```
trace (Math.sin(0));           // 0
trace (Math.sin(Math.PI/2));   // 1.0
```

`sin()` 함수를 `cos()`과 함께 사용하여 원 위의 점의 좌표를 계산할 수 있다. `Math.cos()`에 있는 예제를 참조하기 바란다.

참조

`Math.asin()`, `Math.cos()`, `Math.tan()`

Math.sqrt() 메소드

주어진 숫자의 제곱근을 구하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 `Math.sqrt(x)`

인자

x 음이 아닌 숫자

리턴 값

*x*의 제곱근. *x*가 0보다 작으면 NaN이 리턴된다.

설명

`sqrt()` 메소드에서는 피연산자의 양의 제곱근을 리턴한다(물론 수학적으로는 음의 제곱근도 있지만 그 값은 무시한다). `sqrt()` 메소드를 사용하는 것은 다음처럼 `pow()` 메소드를 사용하는 것과 같다.

```
Math.pow(x, 0.5)
```

예제

```
Math.sqrt(4);    // -2도 제곱근이긴 하지만 2만 리턴한다.
Math.sqrt(36);   // -6도 제곱근이긴 하지만 6만 리턴한다.
Math.sqrt(-20);  // NaN을 리턴한다.
```

참조

Math.pow(), Math.SQRT2, Math.SQRT1_2

Math.SQRT1_2 속성

2의 제곱근의 역수. 대략 0.70711 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.SQRT1_2

설명

SQRT1_2 속성은 1/Math.SQRT2 값(2의 제곱근의 역수)과 같은 상수로 그 값은 대략 0.70711 정도이다.

참조

Math.SQRT2

Math.SQRT2 속성

2의 제곱근. 대략 1.41421 정도

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)

문법 Math.SQRT2

설명

SQRT2 속성은 무리수인 루트 2의 값을 나타내는 상수이다. 그 값은 대략 1.41421 정도이다.

참조

Math.SQRT1_2

Math.tan() 메소드

주어진 각도의 탄젠트 값을 리턴하는 메소드

버전 플래시 5(플래시 4 무비로 저장할 때도 사용할 수 있음)**문법** `Math.tan(theta)`**인자****theta** $-\pi/2$ 와 $\pi/2$ 사이의 각도(라디안 단위)**리턴 값**

theta의 탄젠트 값(결과는 -Infinity에서 Infinity 사이의 숫자)

설명

`tan()` 메소드에서는 주어진 각도의 탄젠트 값을 리턴한다. 직각 삼각형에서 탄젠트 값은 주어진 각의 맞은 편에 있는 변의 길이를 그 각에 붙어있는 변의 길이로 나눈 값이다. `Math.sin(theta)/Math.cos(theta)` 값과 같기 때문에, `cos(theta)`가 0에 가까워지면 `tan(theta)`는 Infinity가 된다. 따라서 $-\pi/2$, $\pi/2$, $-3\pi/2$, $3\pi/2$ 와 같은 값에 대해서는 `tan(theta)`를 계산할 수 없다.

예제

```
trace (Math.tan(0));           // 0이 출력된다.
trace (Math.tan(Math.PI/4));   // 1이 출력된다.
```

참조`Math.atan()`, `Math.cos()`, `Math.sin()`

maxscroll 속성텍스트 필드의 첫 줄로 사용할 수 있는
가장 큰 줄의 번호**버전** 플래시 4 이후**문법** `textField.maxscroll`**리턴 값**

텍스트 필드의 첫 줄로 사용할 수 있는 가장 큰 줄의 행번호를 나타내는 양의 정수

설명

maxscroll 속성을 이용하면 텍스트 필드의 scroll 값으로 사용할 수 있는 최대값을 구할 수 있다. 이 값은 화면에 표시되는 영역의 첫 줄을 나타내는 최대 행번호를 나타낸다.

maxscroll 속성을 scroll 속성과 함께 이용하면 스크롤되는 텍스트 필드를 관리할 수 있다.

참조

scroll; 18장의 'maxscroll 속성'

Mouse 객체

마우스 포인터를 화면에 표시하거나
화면에서 숨기기 위한 객체

버전 플래시 5

문법 `Mouse.methodName`

메소드

hide() 마우스 포인터를 숨기는 메소드

show() 마우스 포인터를 화면에 표시하는 메소드

설명

Mouse 객체에는 속성은 없고 메소드만 두 개 있다. Mouse.hide()는 시스템 마우스 포인터를 숨기는 데 쓰이며 보통 사용자 정의 마우스 포인터를 만들 때 사용한다. 또한 풀스크린 상태에서 키보드만으로 제어하는 무비나 터치스크린 장치에서는 마우스 포인터를 숨기는 것이 좋다.

Mouse 객체에서는 마우스 포인터의 위치를 알 수 없다. 마우스 포인터의 위치는 각 무비 클립 객체에 포함된 _xmouse와 _ymouse 속성에 저장된다. 메인 스테이지에서의 마우스 포인터의 위치는 _root._xmouse와 _root._ymouse 속성을 이용하여 알아낼 수 있다.

참조

MovieClip._xmouse, MovieClip._ymouse

Mouse.hide() 메소드

마우스 포인터를 숨기는 메소드

버전 플래시 5**문법** `Mouse.hide()`

설명

`hide()` 메소드는 마우스가 플레이어 위에 있을 때 일반적인 마우스 포인터(보통 화살표 모양)를 화면에 표시되지 않도록 해준다. 마우스가 플래시 플레이어의 활성 스테이지 영역을 벗어나면 시스템 마우스 포인터가 다시 화면에 나타난다.

주의 사항

플래시 5에서는 `Mouse.hide()`를 호출하더라도 마우스 포인터가 텍스트 필드 위를 지날 때는 시스템 텍스트 커서인 I자 모양의 막대가 화면에 표시된다.

예제

`hide()` 메소드는 주로 사용자 정의 포인터를 사용할 때 기본 시스템 마우스 포인터를 숨기기 위해 쓰인다. 플래시에서 사용자 정의 포인터는 마우스를 따라 다니는 무비 클립에 지나지 않는다. `mouseMove` 이벤트를 이용하면 프레임이 바뀔 때마다 클립의 `_x`, `_y` 속성을 업데이트하여 무비 클립이 마우스를 쫓아다니도록 만들 수 있다.

```
// 사용자 정의 포인터 역할을 하는 클립에 들어갈 코드
onClipEvent (load) {
    Mouse.hide();
}

onClipEvent (mouseMove) {
    _x = _root._xmouse;
    _y = _root._ymouse;
    updateAfterEvent();
}
```

마우스가 비활성화되어 있을 때는 사용자 정의 포인터를 숨기는 것이 좋다. 포인터가 플래시 플레이어의 활성 스테이지 영역을 벗어나면 시스템 포인터와 사용자 정의 포인터가 한꺼번에 화면에 나타날 수 있기 때문이다. 다음과 같은 코드를 이용하면 사용자 정의 포인터가 비활성화되고 나서 5초가 지나면 커서를 숨기도록 할 수 있다.

```

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

onClipEvent (enterFrame) {
    if (getTimer() - lastMove > 5000) {
        _visible = false;
    } else {
        _visible = true;
    }
}

onClipEvent (mouseMove) {
    _x = _root._xmouse;
    _y = _root._ymouse;
    lastMove = getTimer();
    updateAfterEvent();
}

```

마우스가 어떤 버튼을 지나갈 때 사용자 정의 포인터를 사용자 정의 “hand” 아이콘으로 바꾸려면 다음과 같이 버튼의 rollOver 이벤트를 이용하여 사용자 정의 포인터 클립을 hand 아이콘으로 바꾸고 rollOut 이벤트를 이용하여 원래 아이콘으로 바꾸면 된다.

```

on (rollOver) {
    _root.customPointer.gotoAndStop("hand");
}

on (rollOut) {
    _root.customPointer.gotoAndStop("default");
}

```

이러한 작업을 할 때는 사용자 정의 포인터를 무비의 최상위 레이어에 놓아 다른 화면보다 위로 갈 수 있도록 만들어야 한다. duplicateMovieClip()이나 attachMovie() 함수를 이용하여 동적으로 사용자 정의 포인터 클립을 만들고 depth 값을 높게 설정해도 된다.

참조

Mouse.show(), MovieClip._xmouse, MovieClip._ymouse

Mouse.show() 메소드

마우스 포인터를 화면에 표시하는 메소드

버전 플래시 5**문법** `Mouse.show()`**설명**

`show()` 메소드는 `Mouse.hide()`를 호출한 다음, 기본 시스템 마우스 포인터를 화면에 다시 표시하는 데 쓰인다. 폼에서 사용자 입력을 받는 것과 같이 무비에서 일반적인 포인터를 표시해야 하는 경우에 이 메소드를 사용한다. 시스템 포인터가 숨겨져 있을 때 I자 모양의 포인터만 나타나는 경우를 처리할 때도 이 메소드를 사용할 수 있다.

참조`Mouse.hide()`

Movieclip 클래스

메인 무비와 무비 클립을 위한 클래스와 비슷한데이터형

버전 플래시 3 이후**생성자**

없음. 무비 클립 심벌은 플래시 저작 도구에서 수동으로 만들어야 한다. 무비 클립 인스턴스는 `attachMovie()`나 `duplicateMovieClip()` 함수로 만들 수 있다.

속성

무비 클립 속성은 무비 클립과 메인 무비의 다양한 물리적 특징을 기술하거나 제어하는 데 쓰인다. 속성은 다른 객체와 마찬가지로 점 연산자를 이용하여 액세스할 수 있다. 무비 클립 속성의 사용법은 '13장. 무비 클립'의 '무비 클립의 객체성'에 자세히 나와 있다.

속성과 관련하여 다음과 같은 점에 주의해야 한다.

- `MovieClip` 객체의 물리적 속성을 변경하면 그 클립을 프로그램을 통해 제어하게 되므로 내부 타임라인에서 그 클립을 제어할 수 없다. 따라서 현재 내부에서 연결된 부분이 모두 끊어진다.

- MovieClip 속성 중에는 부동소수점 소수처럼 보이긴 하지만 내부적으로는 다른 형식으로 매핑되는 경우도 있다. 예를 들어 `_alpha` 속성은 내부적으로 0 이상 255 이하의 정수로 연결된다. 따라서 어떤 속성을 설정하고 그 값을 확인할 때 정확도가 떨어질 수도 있다. 예를 들어 어떤 값을 90으로 설정하고 나서 바로 그 값을 확인했을 때 89.84375라는 값이 나오는 경우도 있다. 속성 값이 약간 변하더라도 별로 문제가 되지 않는다면 `Math.round()`, `Math.floor()`, `Math.ceil()` 등의 메소드를 이용하고, 정확한 값이 필요하다면 원본 값을 속성이 아닌 다른 변수에 저장해 두는 것이 좋다.

[표 R-9]에 무비 클립 속성의 목록이 나와있다. 모든 내장 무비 클립 속성은 밑줄로 시작한다. 이렇게 하면 사용자 정의 속성(일반적으로 밑줄로 시작하지 않음)과 내장 속성을 구분하기가 좋기 때문이다. [표 R-9]의 액세스 열에 R/W라고 되어 있으면 읽기/쓰기가 모두 가능한 속성, RO라고 되어 있으면 읽기 전용 속성이다. 읽기 전용 속성 중에는 저작 도구나 관련 함수(예: `gotoAndPlay()`를 이용하여 `_currentframe` 속성을 바꿀 수 있음)를 통해 간접적으로 설정할 수 있는 것도 있지만, 액션스크립트를 통해 직접 설정할 수 있는 속성은 읽기/쓰기가 모두 가능한 속성으로 제한된다. 유형 열에는 각 속성 값의 데이터형이 나와있다. 속성 설명 열에는 각 속성의 용도가 간략하게 나와 있으며 자세한 설명은 뒤에 각 속성별로 설명하는 부분에 나와 있다. `_name`과 `_parent` 속성을 제외한 [표 R-9]에 나온 모든 속성은 무비 클립 인스턴스와 메인 무비(즉 `_root`)에 모두 적용된다. 하지만 속성 값은 무비 클립 인스턴스인지 아니면 메인 무비인지에 따라 크게 달라질 수 있으며, 클립이 프로그램을 통해 추가된 클립인 경우에도 현저하게 달라질 수 있다. 예를 들어 어떤 무비 클립의 `_x` 속성은 메인 타임라인의 클립인지 아니면 부모 클립에 속하는 클립인지에 따라 달라진다. 현재 타임라인의 모든 속성은 레퍼런스를 적지 않아도 액세스할 수 있다(예: `_alpha`와 `myClip._alpha`).

[표 R-9] 무비 클립 속성

속성 이름	액세스	유형	속성 설명
<code>_alpha</code>	R/W	number	투명도 백분율: 0은 투명, 100은 불투명
<code>_currentframe</code>	RO	number	플레이헤드가 있는 프레임 번호
<code>_droptarget</code>	RO	string	드래그된 클립이 지나가거나 그 클립을 놓을 대상 클립의 경로. 슬래시 표기법으로 적어야 한다.

속성 이름	액세스	유형	속성 설명
_framesloaded	RO	number	다운로드된 프레임의 개수
_height	R/W	number	현재 콘텐츠의 높이(픽셀 단위)
_name*	R/W	string	문자열(레퍼런스가 아닌) 인스턴스 지시자
_parent*	RO	MovieClip 레퍼런스	현재 인스턴스를 포함하고 있는 인스턴스나 무비의 레퍼런스
_rotation	R/W	number	회전 각도
_target	RO	string	대상의 절대 경로. 슬래시 표기법 사용
_totalframes	RO	number	타임라인에 있는 프레임의 개수
_url	RO	string	원본 .swf 파일의 디스크 또는 네트워크에서의 위치
_visible	R/W	boolean	가시성: true이면 화면에 보이고 false이면 화면에 보이지 않는다.
_width	R/W	number	현재 콘텐츠의 너비(픽셀 단위)
_x	R/W	number	수평 위치(픽셀 단위)
_xmouse	RO	number	마우스 포인터의 수평 위치(픽셀 단위)
_xscale	R/W	number	수평 방향 배율(퍼센트)
_y	R/W	number	수직 위치(픽셀 단위)
_ymouse	RO	number	마우스 포인터의 수직 위치(픽셀 단위)
_yscale	R/W	number	수직 방향 배율(백분율)

*무비 클립 인스턴스에만 적용되며 메인 타임라인에는 적용되지 않는다.

메소드

무비 클립 메소드는 모든 무비 클립 인스턴스 및 거의 모든 메인 무비 타임라인에서 호출할 수 있다. 꽤 많은 무비 클립 메소드는 같은 이름을 가진 전역 함수와 같은 기능을 제공하면서 `MovieClip.method()`처럼 점 표기법을 통해 사용된다. 같은 이름의 전역 함수가 없는 메소드를 사용할 때는 현재 클립의 레퍼런스를 따로 적어 주지 않아도 된다(예를 들면 `attachMovie()`와 `myClip.attachMovie()`는 똑같다). [표 R-10]에 무비 클립 메소드 목록을 수록하였다.

[표 R-10] 무비 클립 메소드

메소드 이름	메소드 설명
attachMovie()	현재 문서의 라이브러리에 있는 심벌로 새로운 인스턴스를 만든다. 새로운 인스턴스는 호스트 클립 또는 프로그램을 통해 만든 무비의 클립 스택에 들어간다.
duplicateMovieClip()*	현재 인스턴스의 복사본을 만들고 그 복사본을 프로그램으로 만든 클립 스택에 집어넣는다(13장의 'duplicateMovieClip()을 이용하여 만드는 방법' 참조).
getBounds()	클립이 화면에 표시되는 영역을 정의하는 경계상자의 좌표가 속성으로 저장된 객체를 리턴한다.
getBytesLoaded()	인스턴스 또는 무비를 다운로드할 때 다운로드된 바이트 수를 리턴한다. 내부 클립에서는 사용할 수 없다.
getBytesTotal()	인스턴스 또는 메인 물리적인 무비의 바이트 크기를 리턴한다.
getURL()	외부 문서(보통 웹 페이지)를 브라우저로 불러온다.
globalToLocal()	coordinates 객체의 속성을 스테이지 좌표에서 인스턴스 좌표로 변환한다. 메인 무비 객체(_root)에 대해 호출하면 원본과 대상 좌표계가 똑같기 때문에 내용이 바뀌지 않는다.
gotoAndPlay()	인스턴스 또는 무비의 플레이헤드를 특정 프레임으로 옮기고 나서 그 인스턴스 또는 무비를 재생한다.
gotoAndStop()	인스턴스 또는 무비의 플레이헤드를 특정 프레임으로 옮기고 플레이헤드를 멈춘다.
hitTest()	어떤 클립이 주어진 점이나 다른 클립과 교차되는지를 나타내는 부울 값을 리턴한다.
loadMovie()	외부 .swf 파일을 플레이어로 불러온다.
loadVariables()	변수 이름과 값의 쌍으로 이루어진 외부 데이터를 가져와서 액션스크립트 변수로 변환한다.
localToGlobal()	coordinates 객체의 속성을 인스턴스 좌표에서 메인 무비 좌표로 변환한다.
nextFrame()	인스턴스 또는 무비의 플레이헤드를 다음 프레임으로 옮기고 나서 멈춘다.
play()	인스턴스나 무비의 플레이헤드를 작동시킨다(클립을 재생한다).
prevFrame()	인스턴스 또는 무비의 플레이헤드를 이전 프레임으로 옮기고 나서 멈춘다.
removeMovieClip()*	복사 또는 추가한 인스턴스를 지운다.
startDrag()	인스턴스나 무비가 마우스 포인터를 따라 움직이도록 만든다.
stop()	인스턴스 또는 무비의 재생을 멈춘다.

메소드 이름	메소드 설명
stopDrag()	현재 진행중인 드래그 작업(마우스 포인터를 따라 움직이는 작업)을 모두 멈춘다.
swapDepths()*	인스턴스 스택에서 인스턴스의 위치를 변경한다.
unloadMovie()	문서 레벨 또는 호스트 클립에서 인스턴스나 메인 무비를 제거한다.
valueOf()	점 표기법을 이용하여 인스턴스의 절대 경로를 나타낸다.

* 무비 클립에만 적용되며 메인 타임라인에는 적용되지 않는다.

이벤트

무비 클립 인스턴스는 미리 정의된 이벤트(마우스나 키보드 상호 작용 또는 무비 재생 등)에 자동으로 반응하는 이벤트 핸들러를 지원한다. [표 R-11]은 플래시에서 지원되는 무비 클립 이벤트 핸들러의 목록이다. 각 핸들러에 대한 자세한 내용은 '10장. 이벤트 및 이벤트 핸들러'를 참조하기 바란다.

[표 R-11] 무비 클립 이벤트 핸들러

클립 이벤트 핸들러	클립 이벤트가 일어나는 경우
onClipEvent (enterFrame)	플래시 플레이어에서 어떤 프레임이 시작될 때
onClipEvent (load)	클립이 스테이지에 처음 나타날 때
onClipEvent (unload)	클립이 스테이지에서 제거될 때
onClipEvent (data)	외부에서 불러온 변수나 로딩하는 무비의 일부를 전송하는 스트림의 마지막 부분을 받았을 때
onClipEvent (mouseDown)	클립이 스테이지에 있는 상태에서 주 마우스 버튼을 눌렀을 때
onClipEvent (mouseUp)	클립이 스테이지에 있는 상태에서 주 마우스 버튼을 눌렀다가 떼일 때
onClipEvent (mouseMove)	클립이 스테이지에 있는 상태에서 마우스 포인터를 움직일 때
onClipEvent (keyDown)	클립이 스테이지에 있는 상태에서 키를 누를 때
onClipEvent (keyUp)	클립이 스테이지에 있는 상태에서 키를 눌렀다가 떼일 때

설명

MovieClip은 사실 액션스크립트 클래스의 일종이 아니고 무비 클립을 나타내고 제어하기 위한 별도의 액션스크립트 데이터형이다. 대부분의 경우에는 무비 클립과 무비를 객체로 간주하여 무비 클립 속성이나 메소드를 만들거나 액세스하면 된다.

MovieClip은 진정한 클래스가 아니기 때문에 새로운 MovieClip 인스턴스는 생성자를 이용하여 만들 수 없다. 대신 저작 도구에서 무비 클립 심벌을 만들고 수동으로 인스턴스를 스테이지에 추가해야 한다. 하지만 메소드를 이용하여 원래 있던 클립을 복사(duplicateMovieClip())하거나 프로그램을 통해 새로운 클립을 스테이지에 추가(attachMovie())할 수도 있다.

모든 MovieClip 객체가 동등한 것은 아니다. MovieClip 메소드와 속성 중에는 메인 무비에서는 사용할 수 없고 무비 클립 인스턴스에서만 사용할 수 있는 것도 있다(메인 무비는 .swf 문서의 _root 타임라인이다). MovieClip 속성과 메소드를 살펴보다 보면 한쪽에만 해당하는 경우가 있다. 여기서 MovieClip은 객체의 종류를, 무비클립은 액션스크립트 데이터형을 의미하며, 무비 클립, 클립 또는 인스턴스는 특정 무비 클립을, 무비는 .swf 파일의 메인 무비를 의미한다. 각 속성과 메소드를 설명할 때 mc라고 적은 것은 클립이나 메인 무비의 이름을 나타낸다. 많은 속성과 메소드에서 mc는 필수적인 것이 아니며, mc를 생략하면 현재 타임라인을 기본값으로 사용한다.

MovieClip과 관련된 부분에서 좌표에 대해 얘기할 때는 무비 클립의 위치를 알아야 한다. 클립의 위치는 클립의 라이브러리 심벌에서 조그마한 십자가 모양으로 표시된 등록 지점(registration point)을 기준으로 나타낸다.

클립이 메인 스테이지에 있으면 스테이지의 왼쪽 위 구석((0,0) 지점)을 기준으로 위치를 따진다. 클립이 다른 클립에 포함된 경우에는 그 위치를 부모 클립의 등록 지점을 기준으로 표시하며, 부모 클립의 등록 지점의 좌표가 (0,0)이 된다. 두 경우 모두 (0,0)은 클립의 위치를 나타내는 데 쓰이는 좌표 공간의 원점이 된다. localToGlobal()과 globalToLocal() 메소드를 이용하면 이 두 좌표 공간을 서로 변환할 수 있다.



플레이시의 좌표계는 Y축이 뒤집힌 직각 좌표계이다. 즉 y 값이 아래로 갈수록 감소하지 않고 증가한다. 예를 들어 y 좌표가 100이라면 X축 아래쪽으로 100픽셀 내려가 있는 점이 된다.

클립을 움직이거나 클립의 위치를 알아내거나 다른 객체나 점과의 교차 여부를 알아낼 때 좌표와 관련된 속성과 메소드를 사용한다. 다른 객체나 점과 교차하는지를 알아내는 방법은 클립을 움직일 때 다른 객체에 부딪친 경우에 운동 방향을 바꿔

줘야 할 경우에 많이 쓰이기 때문에, ‘충돌 감지(collision detection)’라고도 부른다 (MovieClip.hitTest() 메소드 참조).

액션스크립트에는 한 점(x와 y 좌표값 등)을 나타내기 위한 기본 데이터형이 내장되어 있지 않다. 범용 객체로부터 점 객체를 만드는 방법은 MovieClip.globalToLocal()을 설명하는 부분에 나와 있다.

참조

MovieClip 클래스에 대한 전반적인 내용은 13장을 참조하기 바란다.

MovieClip._alpha 속성

클립이나 무비의 투명도

버전 플래시 4 이후

문법 mc._alpha

엑세스 읽기/쓰기

설명

_alpha 속성은 부동소수점 소수로 표현되며 mc의 투명도를 백분율로 나타내는 값이다(0은 완전히 투명한 것, 100은 완전히 불투명한 것을 나타냄). mc의 _alpha 속성에 어떤 값을 대입하면 mc에 포함된 모든 클립의 투명도가 바뀌긴 하지만, 각 중첩 객체의 _alpha 값까지 바뀌는 것은 아니다. 즉 circle이라는 클립이 square에 포함되어 있을 때 square._alpha를 50으로 설정하면 circle이 화면에서 50% 투명하게 표시되긴 하지만, circle._alpha 값은 처음에 설정된 값에서 바뀌지 않는다.

주의 사항

무비 클립의 _alpha 값을 바꾸면 Color.getTransform() 메소드에서 리턴하는 aa 속성도 바뀐다.

예제

```
ball._alpha = 60;    // ball 객체를 반투명하게 만든다.
ball._alpha = 0;     // ball 객체가 아예 보이지 않는다(완전 투명).
```

다음과 같은 클립 이벤트 핸들러를 이용하면 마우스가 화면 아래로 움직일수록 클립을 더 투명하게 보이도록 만들 수 있다.

```
onClipEvent(enterFrame) {
    _alpha = 100 - (_root._ymouse / 400) * 100;
}
```

참조

Color 클래스, MovieClip_visible

MovieClip.attachMovie() 메소드

라이브러리 심벌로부터 새로운 무비 클립 인스턴스를 만드는 메소드

버전 플래시 5

문법 mc.attachMovie(symbolIdentifier, newName, depth)

인자

symbolIdentifier

라이브러리의 Options → Linkage에서 설정한 무비 클립 심벌의 연결 인식자를 나타내는 문자열

newName 새로 만드는 클립의 이름을 나타내는 문자열. 이 이름은 '14장. 렉시컬 구조'의 '인식자' 부분에서 설명한 규칙에 따라 만들어야 한다.

depth

프로그램으로 만든 클립 스택에서 mc 위에 새로운 클립을 놓을 레벨을 나타내는 정수. -1은 0 밑에, 1은 0 앞에, 2는 1 앞으로 가는 식으로 올라간다. 자세한 내용은 13장의 'attachMovie()'를 이용하여 생성된 클립이 스택에 추가되는 과정'을 참조하기 바란다. depth를 음수로 지정해도 별 문제 없이 작동하긴 하지만 액션스크립트에서 음수 depth를 공식적으로 지원하지는 않기 때문에, 나중에 depth를 양수만으로 사용할 수 있도록 되었을 때의 호환성 문제를 감안하여 0 이상의 값만 사용하는 것이 좋다.

설명

`attachMovie()` 메소드는 `symbolIdentifier`로 지정한 무비 클립 심벌을 기반으로 `newName`이라는 새로운 인스턴스를 만든다. `mc`가 메인 무비이면 새로운 인스턴스는 스테이지의 왼쪽 맨 위에 표시된다. `mc`가 무비 클립 인스턴스이면 새로운 인스턴스는 `mc`의 등록 지점에 놓인다. 어떤 경우라도 새로운 인스턴스는 프로그램으로 생성된 클립의 스택에서 `mc` 위로 올라간다.

참조

`duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`; 13장의 ‘`attachMovie()`를 이용하여 만드는 방법’, ‘`attachMovie()`를 이용하여 생성된 클립이 스택에 추가되는 과정’

MovieClip._currentframe 속성

클립이나 무비의 플레이헤드의 프레임 번호

버전	플래시 4 이후
문법	<code>mc._currentframe</code>
액세스	읽기 전용

설명

정수 속성인 `_currentframe`은 `mc`의 플레이헤드가 있는 프레임의 번호를 나타낸다. 첫 번째 프레임은 0이 아니라 1이기 때문에, `_currentframe`의 범위는 1에서 `mc._totalframes`까지이다.

예제

```
// 플레이헤드를 현위치에서 두 프레임 앞으로 움직인다.  
gotoAndStop(_currentframe - 2);
```

참조

`MovieClip.gotoAndPlay()`, `MovieClip.gotoAndStop()`

MovieClip._droptarget 속성

드래그한 클립을 놓을 클립이나 무비의 경로

버전	플래시 4 이후
문법	<code>mc._droptarget</code>
액세스	읽기 전용

설명

mc를 드래그할 때 `_droptarget`에는 mc가 지나가는 클립의 경로를 나타내는 문자열이 저장된다. mc가 어떤 클립 위로도 지나가지 않으면 `_droptarget`에는 `undefined`가 저장된다. 전에 mc를 드래그하여 어떤 클립 위에 놓은 적이 있다면, `_droptarget` 속성에 그 클립에 대한 경로를 나타내는 문자열이 저장된다. 이 경로는 슬래시 표기법으로 표현된다. 만약 드래그한 클립의 등록 지점이 대상 클립의 일부와 겹치면 드래그한 클립이 대상 클립 위에 있는 것으로 간주된다.

예제

`_droptarget` 속성은 드래그-앤-드롭 인터페이스를 만들 때 유용하다. 아래 예제에는 `_droptarget`을 이용하여 간단한 장바구니 인터페이스를 만드는 방법이 나와 있다(item을 basket 클립에 떨어뜨리면 item이 basket 위에 놓이고, 그렇지 않으면 원래 위치로 돌아간다).

```
// item의 1번 프레임에 들어갈 코드
var origX = _x;
var origY = _y;

function drag() {
    this.startDrag();
}

function drop() {
    stopDrag();
    if (_droptarget != "/basket") {
        _x = origX;
        _y = origY;
    }
}
```



```
// item에 있는 버튼에 들어갈 코드
on (press) {
    drag();
}

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

`_droptarget`에는 클립 레퍼런스가 아니라 문자열이 저장된다는 점에 주의하자. `_droptarget` 문자열을 클립 레퍼런스로 변환할 때는 `eval()` 부분의 예제에 나온 방법을 이용하면 된다.

참조

`MovieClip.startDrag()`, `MovieClip.stopDrag()`

MovieClip.duplicateMovieClip() 메소드

무비 클립의 복사본을 만든다.

버전	플래시 5부터 메소드 형태로도 쓰임(전역 함수는 플래시 4부터 쓰임)
문법	<code>mc.duplicateMovieClip(<i>newName</i>, <i>depth</i>)</code>
인자	
<i>newName</i>	복사된 클립의 인스턴스 이름으로 사용할 문자열. 이 이름은 14장의 '인식자' 절에서 설명한 규칙대로 만들어야 한다.
<i>depth</i>	프로그램으로 만든 클립 스택에서 mc 위에 새로운 클립을 놓을 레벨을 나타내는 정수. -1은 0 밑에, 1은 0 앞에, 2는 1 앞으로 가는 식으로 올라간다. 자세한 내용은 13장의 'duplicateMovieClip()'를 이용하여 생성된 클립이 스택에 추가되는 과정'을 참조하기 바란다. <code>depth</code> 를 음수로 지정해도 별 문제 없이 작동하긴 하지만 액션스크립트에서 음수 <code>depth</code> 를 공식적으로 지원하지 않기 때문에, 나중에 <code>depth</code> 를 양수만으로 사용할 수 있도록 되었을 때의 호환성 문제를 감안하여 0 이상의 값만 사용하는 것이 좋다.

설명

MovieClip.duplicateMovieClip() 메소드는 전역 함수인 duplicateMovieClip() 대신 사용할 수 있는 메소드이다. MovieClip의 메소드 형식으로 호출하면 target 매개변수를 지정하지 않아도 되고, mc 클립을 자동으로 복사한다. 전역 함수에 비하면 메소드로 사용하는 것이 사용자의 실수로 인한 오류를 방지하는 데 좋다.

주의 사항은 duplicateMovieClip() 전역 함수를 설명하는 부분에 나와 있다.

참조

MovieClip.attachMovie(), duplicateMovieClip()

MovieClip._framesloaded 속성

플레이어에 다운로드된 클립이나 무비의 프레임 수

버전 플래시 4 이후

문법 mc._framesloaded

액세스 읽기 전용

설명

_framesloaded 속성은 mc를 플레이어에 다운로드할 때 다운로드된 프레임 수를 나타낸다(범위는 0부터 mc._totalframes까지). 일반적으로 프레임이 충분히 다운로드될 때까지 무비의 재생을 멈출 수 있도록 프리로더를 만들 때 이 속성을 이용한다. 무비 클립의 경우에는 loadMovie()를 호출하여 인스턴스에서 외부 .swf 파일을 로딩하는 경우를 제외하면 _framesloaded 속성이 언제나 _totalframes와 같은 값을 가진다(재생을 시작하기 전에 클립 전체를 다운로드하기 때문이다). 따라서 _framesloaded 속성은 인스턴스나 레벨에 외부 .swf 파일을 불러오는 경우, 또는 메인 무비를 로딩하는 경우에만 제대로 사용할 수 있다.

프리로더 코드는 로딩중인 무비의 메인 타임라인에 직접 집어넣는 방식으로 만든다. 가장 간단한 방법은 무비가 모두 로딩될 때까지 1번과 2번 프레임 사이에서 루프를 돌리는 방법이다. 무비의 로딩이 완료되면 무비의 시작 프레임으로 이동한다.

```
// 1번 프레임의 코드
if (_framesloaded > 0 && _framesloaded == _totalframes) {
    gotoAndPlay("beginMovie");
}

// 2번 프레임의 코드
gotoAndPlay(1);
```

플레이시 5 이후에서는 enterFrame 무비 클립 이벤트 핸들러를 이용하여 더 간편하게 프리로더를 만들 수 있다. 우선 로딩하고자 하는 무비의 프리로딩을 시작할 프레임에서 stop() 함수를 호출한다. 그리고 나서 무비의 타임라인에 다음과 같은 코드를 추가한다.

```
onClipEvent (enterFrame) {
    loaded = _parent._framesloaded;
    if (loaded > 0 && loaded == _parent._totalframes && loading !=
"done") {
        _parent.gotoAndPlay("beginMovie");
        loading = "done";
    }
}
```

위 예제에 있는 클립에서는 부모의 로딩 상황을 추적하고 로딩이 완료되면 부모 클립을 재생시킨다. 무비 클립을 프리로더로 이용하면 로딩 중인 무비의 타임라인에 루프를 만들지 않아도 된다. 무비 클립 프리로더를 스마트 클립으로 만들어 초보 개발자들이 손쉽게 프리로더를 구현할 수 있도록 할 수도 있다.

앞에서 만든 프리로더 예제에서는 _framesloaded == _totalframes 외에도 _framesloaded > 0도 확인한다. 이렇게 하는 이유는 무비를 클립에서 언로드할 때는 그 클립의 _totalframes가 0이 되기 때문이다. 따라서 _framesloaded가 0이면(인터넷 접속 속도가 매우 느리다면 그럴 수도 있다) 아직 한 프레임도 로딩되지 않은 경우에도 _framesloaded == _totalframes가 true가 될 수도 있다. 따라서 미리 _framesloaded > 0을 체크해 두면 필요한 내용이 모두 로딩되기 전에 무비 내용을 건너뛰는 사고를 방지할 수 있다. 어떤 레벨로 로딩되는 .swf 파일의 메인 무비 프리로더에서는 _totalframes 속성이 절대 0이 되지 않기 때문에 이렇게 하지 않아도 된다.

예제

프리로더를 만들 때는 다운로드된 비율을 나타내는 수평 막대와 텍스트 필드를 사용하는 경우가 많다. 수평 막대를 로딩하는 작업은 `_width` 속성이나 `_xscale` 속성으로 크기를 조절하는 클립으로 구현한다. 클립이 확대되거나 축소될 때는 항상 등록 지점(이 점을 기준으로 오른쪽과 왼쪽으로 확대/축소된다)을 기준으로 움직인다는 점에 주의하자. 따라서 클립을 한 방향으로만 확대하고 싶다면 클립의 내용물을 모두 클립 심벌에 있는 등록 지점을 기준으로 한 방향으로 몰아야 한다. 아래 예는 앞에서 만든 클립 핸들러 코드에 로딩 상태를 확인할 수 있는 수평 막대와 텍스트 필드를 추가한 코드이다.

```
// 상태표시줄이 있는 프리로더
onClipEvent (enterFrame) {
    // 로딩된 프레임 수를 알아낸다.
    loaded = _parent._framesloaded;
    // 모든 프레임을 로딩하고 나면
    if (loaded > 0 && loaded == _parent._totalframes && loading != "done") {
        // 무비 재생을 시작하고 로딩이 끝났음을 표시한다.
        _parent.gotoAndPlay("beginMovie");
        loading = "done";
    }
    // 로딩된 바이트 수의 백분율을 알아낸다.
    percentDone = Math.floor((_parent.getBytesLoaded()
                               / _parent.getBytesTotal()) * 100);
    // loadStatus라는 텍스트 필드에 위에서 구한 값을 표시한다.
    loadStatus = percentDone + "% complete";
    // loadBar 클립의 크기를 설정한다.
    loadBar._xscale = percentDone;
}
```

프리로더를 테스트하기 위해 무비를 다운로드하는 과정을 시뮬레이션하고 싶다면, Test Movie 모드에서 Bandwidth Profiler를 이용하면 된다.

참조

`MovieClip.getBytesLoaded()`, `MovieClip._totalframes`; 10장의 ‘데이터’ 및 19장의 ‘대역폭 프로파일러’

MovieClip.getBounds() 메소드

클립이나 무비의 경계 상자를 구하는 메소드

버전 플래시 5**문법** `mc.getBounds(targetCoordinateSpace)`**인자*****targetCoordinateSpace***

mc의 크기를 측정할 무비나 클립의 경로를 나타내는 문자열. 문자열을 사용할 자리에 무비 클립 레퍼런스를 사용하면 자동으로 경로 형식으로 변환되므로, mc.getBounds(_root)처럼 무비 클립 객체 레퍼런스를 이용한 결과도 mc.getBounds("_root")를 사용한 경우와 똑같다. 아무 값도 지정하지 않으면 mc가 기본값이 된다.

리턴 값

mc가 차지하고 있는 영역의 경계 상자를 나타내는 속성(xMin, xMax, yMin, yMax)이 저장된 객체. 이 객체에 있는 네 가지 속성은 mc의 왼쪽 맨 아래, 오른쪽 맨 위에 있는 두 개의 점을 나타낸다.

설명

getBounds() 메소드에서는 mc가 차지하는 직사각형 영역(즉 mc의 경계 상자)을 정의할 수 있는 값을 속성으로 가지는 익명 객체를 리턴한다. 리턴된 객체에 있는 값을 이용하려면 아래 예제에 나온 방법을 이용하여 객체의 속성을 액세스하면 된다.

클립의 경계 상자의 크기는 다른 임의의 클립이나 무비를 기준으로 측정할 수 있다. targetCoordinateSpace 인자를 이용하면 mc가 targetCoordinateSpace의 캔버스에 들어갔을 때 어느 영역을 차지하는지 알 수 있다. 이 메소드의 결과는 targetCoordinateSpace가 메인 무비인지 아니면 클립 인스턴스인지에 따라 달라진다. 메인 무비 좌표계에서의 원점은 스테이지의 맨 왼쪽 위 지점이지만, 인스턴스 좌표계의 원점은 클립의 라이브러리 심벌에서 십자가 모양으로 표시된 등록 지점이다.

getBounds() 메소드를 이용하면 무비 클립 사이 또는 어떤 점과의 충돌을 감지할 수 있다(물론 이러한 용도로는 MovieClip.hitTest()를 사용하는 것이 좋다). 또한 MovieClip.attachMovie()를 이용하여 클립을 추가할 때 그 클립이 추가되는 직사각형 영역을 알아낼 수도 있다.

예제

```
var clipBounds = this.getBounds();
var leftX      = clipBounds.xMin;
var rightX     = clipBounds.xMax;
var topY       = clipBounds.yMin;
var bottomY    = clipBounds.yMax;
```

참조

MovieClip.hitTest()

MovieClip.getBytesLoaded() 메소드

플레이어에 다운로드된 바이트 수를
확인하는 메소드

버전 플래시 5

문법 `mc.getBytesLoaded()`

리턴 값

mc를 다운로드할 때 플레이어에 다운로드된 바이트 수를 나타내는 정수(킬로바이트 단위로 변환하려면 1024로 나누면 된다)

설명

getBytesLoaded() 메소드를 이용하면 플래시 플레이어에 다운로드된 바이트 수를 알 수 있다. 하지만 getBytesLoaded()에서는 한 프레임이 도착할 때마다 바이트를 계산한다. 어떤 무비의 첫 번째 프레임이 200바이트이고 두 번째 프레임은 3000바이트라면 getBytesLoaded()에서는 200이나 3200만을 리턴할 뿐, 그 사이의 값은 리턴하지 않는다. 즉 한 프레임을 다운로드하는 도중에는 getBytesLoaded()의 리턴 값이 바뀌지 않는다. 따라서 getBytesLoaded() 메소드는 _framesloaded 속성을 바이트 단위로 바꾼 버전이라고 생각하면 된다.

내부 무비 클립은 언제나 화면에 다운로드가 끝난 다음에 화면에 표시되기 때문에 내부 무비 클립에서 getBytesLoaded()를 호출하면 언제나 getBytesTotal()의 리턴 값과 같은 값을 리턴한다(무비 클립에서 loadMovie() 메소드로 외부 .swf 파일을 로딩하는 경우에는 다르다). 따라서 getBytesLoaded()는 인스턴스나 레벨로 외부 .swf 파일을 불러오는 경우 또는 메인 무비를 로딩하는 경우에만 유용하다.

_framesloaded와 마찬가지로 getBytesLoaded()도 프리로더를 만들 때 많이 사용한다. getBytesTotal()과 함께 사용하면 _framesloaded와 _totalframes를 이용한 경우보다 더 정확하게 다운로드 상태를 확인할 수 있다(각 프레임 크기가 다를 수 있기 때문이다. 프레임이 열 개인 무비를 다운로드하는 경우에 후반부의 프레임 크기가 매우 크다면 처음 세 개의 프레임을 다운로드했다고 해서 30% 다운로드했다고 보기는 힘들다).

예제

```
// 1번 프레임의 코드
if (_framesloaded > 0 && _framesloaded == _totalframes) {
    gotoAndPlay("beginMovie");
} else {
    // 텍스트 필드에 다운로드 상태를 표시한다. 1024로 나누어 KB 단위로 표시한다.
    loadProgressOutput = this.getBytesLoaded()/1024;
    loadTotalOutput = this.getBytesTotal()/1024;
}

// 2번 프레임의 코드
gotoAndPlay(1);
```

참조

MovieClip._framesloaded, MovieClip.getBytesTotal()

MovieClip.getBytesTotal() 메소드

클립이나 무비의 크기를 바이트 단위로
리턴하는 메소드

버전 플래시 5

문법 mc.getBytesTotal()

리턴 값

mc의 크기를 바이트 단위로 나타낸 정수. 1024로 나누면 킬로바이트 단위의 값을 알 수 있다.

설명

getBytesTotal() 메소드를 이용하면 클립 인스턴스 또는 메인 무비의 크기를 바이트 단위로 알아낼 수 있다. 메인 무비에 대해 이 메소드를 사용하면 전체 .swf 파일의 크기를 리턴한다. 일반적으로 getBytesLoaded()와 함께 인스턴스나 특정 레벨에 .swf 파일을 로딩하거나 메인 무비를 로딩할 때 사용할 프리로더를 만들기 위한 용도로 쓰인다.

참조

MovieClip.getBytesLoaded(), MovieClip._totalframes

MovieClip.getURL() 메소드

주어진 문서를 브라우저 창으로 불러오는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 2부터 전역 함수 형태로 지원되었으며 플래시 4에서 method 인자를 사용할 수 있도록 개선되었다)

문법 `mc.getURL(URL, window, method)`

인자

URL 불러올 문서 또는 실행시킬 외부 스크립트의 위치를 나타내는 문자열

window 불러온 문서를 표시할 브라우저 창 또는 프레임의 이름을 나타내는 문자열. 사용자 정의 이름을 사용해도 괜찮고, 미리 정의된 프레임 이름인 "_blank", "_parent", "_self", "top" 중 하나를 사용해도 된다(옵션).

method mc에서 외부 스크립트로 변수를 보낼 때 사용할 방법을 지정하는 문자열 리터럴. "GET", "POST" 중 한 값을 사용해야 하며 반드시 문자열 리터럴로 써야 한다(옵션).

설명

MovieClip.getURL() 메소드는 getURL() 전역 함수와 유사한 메소드이다. 메소드 형태는 현재 타임라인 변수가 아닌 특정 클립에 있는 변수를 보내기 위한 용도로 많이 쓰이기 때문에, 문서를 단순히 불러오는 경우가 아닌 데이터를 전송하는 경우에 주로 쓰인다.

참조

일반적인 주의 사항은 `getURL()` 함수 부분 참조

MovieClip.globalToLocal() 메소드

메인 스테이지의 한 점을
클립 좌표로 변환하는 메소드

버전 플래시 5

문법 `mc.globalToLocal(point)`

인자

point 플레이어의 메인 스테이지(_root) 위의 한 점을 나타내는 x와 y 속성을 가지는 객체에 대한 레퍼런스. x와 y는 부동소수점 소수이다.

설명

`globalToLocal()` 메소드는 `point`의 x와 y 속성을 메인 스테이지 좌표계에서 mc의 좌표 공간으로 변환한다. `globalToLocal()`에서는 새로운 객체를 리턴하는 대신 `point`의 x와 y 값을 바꾼다는 점에 주의하자.

`globalToLocal()`을 사용하려면 우선 다음과 같이 x와 y 속성을 가지는 객체를 만들어야 한다.

```
var myPoint = new Object();
myPoint.x = 10;
myPoint.y = 20;
```

이렇게 만든 객체의 x와 y 속성은 메인 스테이지의 왼쪽 위 구석을 기준으로 하는 수평 및 수직 방향의 위치를 나타내는 값이다. 예를 들어 x가 10이면 스테이지의 왼쪽 끝에서 오른쪽으로 10픽셀 떨어진 위치를, y가 20이면 스테이지의 위쪽 끝에서 아래로 20픽셀 떨어진 위치를 나타낸다. 객체를 만들고 x와 y를 설정하고 나면 그 객체를 다음과 같이 `globalToLocal()` 메소드에 전달하면 된다.

```
myClip.globalToLocal(myPoint);
```

`globalToLocal()`이 실행되면 `myPoint`의 `x`와 `y` 속성이 변환되어 `myClip`의 등록 지점으로부터 측정된 좌표로 바뀐다. `myPoint` 객체의 `x`, `y` 속성 값을 다시 확인해보면 메인 스테이지의 (`x`, `y`) 지점이 `myClip`에서는 어떻게 표현될지 알 수 있다.

```
xInClip = myPoint.x;
yInClip = myPoint.y;
```

예제

아래 예제에서는 메인 스테이지의 왼쪽 위 구석을 기준으로 현재 클립의 등록 지점의 오프셋을 알 수 있다.

```
pt = new Object();           // 점을 저장할 범용 객체를 만든다.
pt.x = 0;                    // 메인 스테이지의 왼쪽 경계선
pt.y = 0;                    // 메인 스테이지의 위쪽 경계선
this.globalToLocal(pt);      // pt를 로컬 좌표계로 변환한다.

trace("From the current clip, the top-left corner of the main Stage is at");
trace("x: " + pt.x + "y: " + pt.y);
```

참조

`MovieClip.localToGlobal()`

MovieClip.gotoAndPlay() 메소드 주어진 프레임으로 건너뛴 다음 무비를 재생하는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(전역 함수는 플래시 2부터 지원)

문법 `mc.gotoAndPlay(frameNumber)`
 `mc.gotoAndPlay(frameLabel)`

인자

frameNumber

`mc`의 플레이헤드를 움직여서 무비를 재생할 프레임의 번호를 나타내는 정수. `frameNumber`가 1보다 작거나 `mc`의 타임라인에 있는 프레임 개수보다 크면 플레이헤드는 각각 첫 번째 또는 마지막 프레임으로 이동한다.

frameLabel

mc의 플레이헤드를 움직여서 무비를 재생할 프레임을 가리키는 레이블을 나타내는 문자열. *frameLabel*을 찾을 수 없으면 mc의 타임라인의 첫 번째 프레임으로 플레이헤드를 이동한다.

설명

MovieClip.gotoAndPlay() 메소드는 gotoAndPlay() 전역 함수와 비슷한 기능을 가지는 메소드이다. mc라는 원격 무비 클립 또는 무비를 제어할 때 주로 메소드 형태로 사용한다.

일반적인 주의 사항은 gotoAndPlay() 전역 함수 부분에 나와 있다.

예제

```
// part1 클립을 intro라는 레이블이 붙은 프레임으로 이동하고 part1을 재생한다.  
part1.gotoAndPlay("intro");
```

MovieClip.gotoAndStop() 메소드 주어진 프레임으로 건너뛴 다음 무비 재생을 중단

버전 플래시 5부터 메소드 형태로도 쓰임(전역 함수는 플래시 2부터 지원)

문법 `mc.gotoAndStop(frameNumber)`
 `mc.gotoAndStop(frameLabel)`

인자***frameNumber***

mc의 플레이헤드를 이동시키고 무비 재생을 멈출 프레임 번호를 나타내는 정수. *frameNumber*가 1보다 작거나 mc의 타임라인에 있는 프레임의 개수보다 크면 플레이헤드는 각각 첫 번째 또는 마지막 프레임으로 이동한다.

frameLabel

mc의 플레이헤드를 이동시키고 무비 재생을 멈출 프레임을 가리키는 레이블을 나타내는 문자열. *frameLabel*을 찾을 수 없으면 mc의 타임라인의 첫 번째 프레임으로 플레이헤드를 이동한다.

설명

MovieClip.gotoAndStop() 메소드는 gotoAndStop() 전역 함수와 비슷한 기능을 가지는 메소드이다. mc라는 원격 무비 클립 또는 무비를 제어할 때 주로 메소드 형태로 쓰인다.

일반적인 주의 사항은 gotoAndStop() 전역 함수 부분에 나와 있다.

예제

```
// mainMenu 클립을 6번 프레임으로 보내고 플레이헤드를 정지시킨다.
mainMenu.gotoAndStop(6);
```

MovieClip._height 속성

클립이나 무비의 높이를 픽셀 단위로 나타낸 값

버전 플래시 4(플래시 5에서 개선되었음)

문법 mc._height

액세스 플래시 4에서는 읽기 전용, 플래시 5에서는 읽기/쓰기

설명

_height 속성은 mc의 높이를 픽셀 단위로 나타내는 음이 아닌 숫자이다. mc에 아무런 내용도 없으면 _height는 0이 된다. _height 속성은 클립의 내용을 가장 높은 픽셀과 가장 낮은 픽셀 사이의 거리를 기준으로 측정한 값이며 픽셀 중에 빈 공간이 있더라도 상관없다. 저작 도구 또는 _yscale 값을 통해 클립의 높이가 바뀌면 _height 속성도 바뀐다.

_height 값을 직접 설정하여 무비 클립의 수직 방향 크기를 조절할 수도 있다. _height에 음수를 대입하려면 그 값은 그냥 무시된다. 클립의 _height를 0으로 설정하더라도 클립이 없어지지 않으며 그냥 한 픽셀짜리 수평선이 된다.

메인 무비의 _height 속성(_root._height 또는 _leveln._height)의 값은 저작 도구의 Modify → Movie → Dimensions에서 설정한 스테이지의 높이가 아니라 메인 무비에 있는 내용물의 높이가 된다. 스테이지의 높이를 직접 나타내는 속성은 없다. 만약 스테이지 높이가 필요하다면 별도로 변수를 만들어서 저장해야 한다. 예를 들어 스테이지 높이가 400이라면 다음과 같은 변수를 추가하면 된다.

```
Object.prototype.stageHeight = 400;
```

주의 사항

무비 클립의 높이를 변경하면 모든 선도 높이가 변경된 비율에 맞추어 변경되므로, Stroke 패널에서 설정한 원래의 크기와 다른 크기로 표시된다. 하지만 Stroke 패널에서 Hairline으로 설정한 선은 무비 클립의 크기를 바꾸더라도 그대로 유지된다. 즉 클립의 크기를 조절하더라도 선이 굵어지거나 가늘어지지 않도록 하려면 Hairline으로 설정하면 된다.

예제

```
ball._height = 20;           // ball의 높이를 20으로 설정한다.
ball._height /= 2;           // ball의 높이를 반으로 줄인다.
```

참조

MovieClip._width, MovieClip._yscale

MovieClip.hitTest() 메소드

어떤 점이 클립 내부에 있는지, 또는 두 클립이 서로 교차하는지 체크하는 메소드

버전 플래시 5

문법 `mc.hitTest(x, y, shapeFlag)`
`mc.hitTest(target)`

인자

x 테스트할 점의 수평 방향 위치

y 테스트할 점의 수직 방향 위치

shapeFlag 충돌 테스트를 mc의 경계 상자(false)에 대해 수행할지, mc의 실제 픽셀(true)에 대해 수행할지를 나타내는 부울 값. 이 인자를 지정하지 않으면 false를 기본값으로 사용한다. shapeFlag는 x, y 인자와는 사용할 수 있지만 target 인자와 함께 사용할 수는 없다는 점에 주의하자. 이 인자는 mc가 울퉁불퉁하거나 가운데가 비어있는 모양인 경우에만 유효하며 직사각형 모양인 경우에는 무의미하다.

target mc와의 충돌 여부를 테스트할 무비 클립의 경로를 나타내는 문자열. 무비 클립 레퍼런스를 문자열이 들어갈 자리에 사용하면 경로로 자동

변환되므로 target 자리에 무비 클립 객체 레퍼런스를 사용해도 된다.
즉 mc.hitTest(ball)이나 mc.hitTest("ball")은 모두 똑같은 의미를 가진다.

리턴 값

충돌 감지 테스트 결과를 나타내는 부울 값. 다음과 같은 상황 중 한 가지라도 해당되면 결과는 true이다.

- 메인 스테이지의 (x, y) 지점이 mc가 차지하는 픽셀과 겹치는 경우. mc가 차지하는 픽셀이란 mc에 의해 표현되는 도형이나 텍스트와 같은 시각적인 요소를 포함하는 픽셀을 의미한다.
- shapeFlag 속성이 false이고 메인 스테이지의 (x, y) 지점이 mc의 경계 상자의 임의의 픽셀과 겹치는 경우. mc의 경계 상자는 mc가 차지하는 픽셀을 모두 포함할 수 있는 가장 작은 직사각형이다.
- target의 경계 상자 안에 있는 픽셀이 하나라도 mc의 경계 상자 안에 있는 픽셀과 겹치는 경우

다음 중 한 가지라도 해당되면 결과는 false가 된다.

- shapeFlag 속성이 true이고 메인 스테이지의 (x, y) 지점이 mc가 차지하는 픽셀과 하나도 겹치지 않는 경우. mc가 차지하는 픽셀이란 mc에 의해 표현되는 도형이나 텍스트와 같은 시각적인 요소를 포함하는 픽셀을 의미한다.
- 메인 스테이지의 (x, y) 지점이 mc의 경계 상자의 어떤 점과도 겹치지 않는 경우. mc의 경계 상자는 mc가 차지하는 픽셀을 모두 포함할 수 있는 가장 작은 직사각형이다.
- target의 경계 상자 안에 있는 픽셀이 mc의 경계 상자 안에 있는 픽셀과 전혀 겹치지 않는 경우

설명

hitTest() 메소드는 어떤 무비 클립이나 특정 지점이 mc와 겹치는지(즉 mc와 충돌하는지) 확인하기 위한 메소드이다.

어떤 점이 mc와 겹치는지 확인하고자 하는 경우에는 hitTest()에 확인할 지점의 x, y 좌표(메인 스테이지 기준)를 전달하면 된다. 옵션으로 shapeFlag를 전달할 수

도 있는데, 이 인자는 충돌 테스트를 mc가 차지하는 픽셀에 대해 수행할지 아니면 mc의 경계 상자(mc에 속하는 모든 픽셀을 포함하는 가장 작은 직사각형)에 대해 수행할지를 나타낸다. mc의 실제 픽셀에 대해 조사하면 주어진 (x, y) 지점이 (mc의 경계 상자가 아니라) mc가 차지하는 픽셀인지 알아낼 수 있다.

어떤 무비 클립이 mc와 겹치는지 확인할 때는 hitTest()를 호출할 때 target 인자를 전달하여 조사할 클립을 지정하면 된다. target과 mc 사이의 충돌을 감지할 때는 항상 클립의 경계 상자를 기준으로 한다. hitTest()에서는 픽셀 단위로 클립과 클립의 충돌을 감지할 수 없다. 일일이 픽셀 단위로 충돌을 감지하려면 코드도 복잡해지고 실행 속도도 많이 느려진다. 여러 가지 경우(예를 들어 우주선 게임같은 경우)에서는 실제 픽셀보다는 클립을 둘러싸는 원을 기준으로 충돌을 조사하는 것만으로도 충분하다.

주의 사항

충돌을 확인할 때는 언제나 플레이어의 메인 스테이지에서의 mc의 위치를 기준으로 한다. 따라서 hitTest()를 한 점에 대해 실행시키면 x와 y 인자를 스테이지 좌표로 지정해야만 한다. 클립 좌표를 스테이지 좌표로 변환하는 방법은 MovieClip.localToGlobal()을 설명하는 부분에 나와 있다. 클립끼리 충돌하는 것을 감지할 때 서로 다른 타임라인에 있는 클립의 위치는 자동으로 전역(스테이지) 좌표로 변환된다.

예제

아래 예에서는 hitTest()를 사용하지 않고 수동으로 두 원이 겹치는지 검사하는 방법을 보여준다.

```
// x, y 축에 대해 두 원형 클립의 거리를 조사한다.
var deltaX = clip1._x - clip2._x; // 수평 방향의 거리
var deltaY = clip1._y - clip2._y; // 수직 방향의 거리

// 각 원의 반지름을 적당한 속성으로 저장한다.
var radius1 = clip1._width / 2;
var radius2 = clip2._width / 2;

// 두 원의 중심간의 거리의 제곱이 두 반지름의 제곱의 합
// 이하인 경우에는 두 원형 클립이 겹친다.
if ((deltaX * deltaX) + (deltaY * deltaY)
    <= (radius1 + radius2) * (radius1 + radius2)) {
```

```

        trace("intersecting");
    } else {
        trace("not intersecting");
    }

```

아래 예는 paddle이 ball과 겹치는지 확인하는 예이다.

```

if (paddle.hitTest("ball")) {
    trace("The paddle hit the ball.");
}

```

다음 예제에서는 tractor가 차지하는 픽셀이 마우스 포인터와 겹치는지 확인한다. 포인터의 좌표는 _root(메인 스테이지)를 기준으로 주어진다. 주의하자. shapeFlag를 true로 하면 무비 클립 안의 비어있는 부분도 감지할 수 있다. 예를 들어 tractor에서 트랙터의 유리를 나타내기 위해 비어있는 공간을 집어넣는다면 포인터가 그 유리 위에 있을 때는 hitTest()에서 false가 리턴된다.

```

if (tractor.hitTest(_root._xmouse, _root._ymouse, true)) {
    trace("You're pointing to the tractor.");
}

```

참조

MovieClip.getBounds(), MovieClip.localToGlobal()

MovieClip.loadMovie() 메소드

외부 .swf 파일을 플레이어로 불러오는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(전역 함수는 플래시 4부터 쓰임)

문법

```

mc.loadMovie(URL)
mc.loadMovie(URL, method)

```

인자

URL 불러올 외부 .swf 파일의 위치를 나타내는 문자열

method 변수를 외부 스크립트로 보낼 때 사용할 방법을 나타내는 문자열 리터럴. "GET"이나 "POST" 둘 중 한 가지만을 인자로 사용할 수 있으며 다른 표현식은 사용할 수 없다(옵션).

설명

MovieClip.loadMovie() 메소드는 loadMovie() 전역 함수와 같은 기능을 가지는 메소드이다. MovieClip 메소드로 호출하면 loadMovie()에 target 매개변수를 사용하지 않아도 된다. URL에 있는 .swf 파일을 자동으로 mc로 불러오기 때문이다. loadMovie() 전역 함수를 사용하는 대신 MovieClip 메소드 형태로 사용하면 오류 발생 가능성을 크게 줄일 수 있다.

자세한 내용은 loadMovie() 전역 함수를 설명하는 부분을 참조하기 바란다.

참조

loadMovie(); 13장의 '메소드와 전역 함수가 겹치는 문제'

MovieClip.loadVariables() 메소드

외부 변수를 가져오는 메소드

버전	플래시 5부터 메소드 형태로도 쓰임(전역 함수는 플래시 4부터 쓰임)
문법	<code>mc.loadVariables(URL)</code> <code>mc.loadVariables(URL, method)</code>
인자	
URL	변수를 출력하는 스크립트나 변수가 저장된 텍스트 파일의 경로를 나타내는 문자열
method	mc에서 외부 스크립트로 변수를 보낼 때 사용할 방법을 나타내는 문자열 리터럴. 이 인자를 사용하면 변수를 받기 전에 변수를 보낸다. 이 값을 생략하면 변수를 가져오기만 한다. "GET"이나 "POST" 둘 중 한 가지만을 인자로 사용할 수 있으며 다른 표현식은 사용할 수 없다.

설명

MovieClip.loadVariables() 메소드는 loadVariables() 전역 함수와 거의 같은 기능을 가진다. MovieClip 메소드로 호출하는 경우에는 target 매개변수를 쓰지 않고 URL에 있는 변수를 바로 mc로 가져온다. 이와 같이 메소드 형태로 사용하면 오류 발생 가능성을 크게 줄일 수 있다.

참조

`loadVariables()`; 13장의 ‘메소드와 전역 함수가 겹치는 문제’

MovieClip.localToGlobal() 메소드

클립에 있는 한 점의 좌표를 메인 스테이지의 좌표로 변환한다.

버전 플래시 5

문법 `mc.localToGlobal(point)`

인자

point mc의 좌표 공간에서 한 점을 나타내기 위한 x와 y 값을 속성으로 가지고 있는 객체에 대한 레퍼런스. x와 y 값은 부동소수점 소수 형태이다.

설명

`localToGlobal()` 메소드는 `point`의 x, y 속성을 mc의 좌표 공간에서 주어진 좌표로부터 플레이어의 메인 스테이지의 좌표계로 변환한다. `localToGlobal()`에서는 새로운 객체를 리턴하지 않고 인자로 전달한 `point`의 속성을 직접 변환한다.

`localToGlobal()`을 사용하려면 우선 x와 y 속성을 가지는 `point` 또는 `coordinates` 객체를 만들어야 한다. 다음과 같이 `Object` 클래스로부터 범용 객체를 만들고 x와 y 속성을 집어넣기만 하면 된다.

```
myPoint = new Object();
myPoint.x = 10;
myPoint.y = 20;
```

객체의 x와 y 속성은 mc의 등록 지점(mc의 라이브러리 심벌에서 십자가 모양으로 나오는 부분)을 기준으로 하는 mc 위의 수평 및 수직 방향 위치를 나타내는 값이다. 예를 들어 x가 10이면 mc의 등록 지점에서 오른쪽으로 10픽셀 떨어진 점, y가 20이면 mc의 등록 지점에서 아래로 20픽셀 떨어진 점을 의미한다. 객체를 만들고 x, y 속성을 설정한 다음 `localToGlobal()` 메소드에 그 객체를 인자로 전달한다.

```
myClip.localToGlobal(myPoint);
```

`localToGlobal()`을 실행하면 `myPoint`의 `x`와 `y` 속성 값이 변환되어 스테이지의 왼쪽 위 구석으로부터 측정한 메인 스테이지에서의 위치를 나타내는 값이 된다. `myPoint` 객체의 속성에 저장된 새로운 값을 확인하면 무비 클립에서의 (`x`, `y`) 지점이 메인 스테이지에서 어디에 나타날지 알 수 있다.

```
mainX = myPoint.x;
mainY = myPoint.y;
```

예제

아래 예제는 메인 스테이지를 기준으로 클립의 등록 지점을 알아내는 코드이다.

```
pt = new Object();
pt.x = 0; // 클립의 등록 지점의 수평 위치
pt.y = 0; // 클립의 등록 지점의 수직 위치
this.localToGlobal(pt); // pt를 메인 스테이지 좌표로 변환한다.

trace("On the main Stage, the registration point of the current clip
is at: ");
trace("x: " + pt.x + "y: " + pt.y);
```

`hitTest()`와 같이 좌표를 스테이지 좌표 공간 기준으로 지정해야 하는 경우에는 `localToGlobal()` 메소드를 이용하여 클립 좌표계에서 메인 스테이지 좌표계로 변환할 수 있다. 두 개의 서로 다른 무비 클립에 있는 점들을 같은 공간에서 비교할 때 이 메소드를 이용하면 된다.

참조

`MovieClip.globalToLocal()`, `MovieClip.hitTest()`

MovieClip._name 속성

클립 인스턴스의 인식자를 문자열로 나타낸 값

버전	플래시 4 이후
문법	<code>mc._name</code>
액세스	읽기/쓰기

설명

`_name` 속성은 무비 클립 인스턴스의 이름을 문자열로 표현한 값이다. 무비를 만들 때 Instance 패널에서 설정한 인스턴스 이름이 `_name` 속성에 저장되기 때문에 메인 무비에서는 이 속성을 사용할 수 없다(메인 무비는 전역 속성인 `_root`로 참조하면 된다).

예제

[예제 10-2]에서 별 모양의 무비 클립을 여러 개 만들었던 것처럼 `_name`을 이용하여 주어진 클립을 조작할지 여부를 결정할 수 있다.

다음과 같이 `_name` 속성을 이용하여 클립의 인식자를 새로 대입할 수 있다.

```
// ball의 이름을 circle로 바꾼다.
ball._name = "circle";

// 바뀐 이름을 이용하여 클립을 조작한다.
circle._x = 500;
```

참조

`MovieClip._target`, `targetPath()`; 13장의 '인스턴스 이름'

MovieClip.nextFrame() 메소드

클립이나 무비의 플레이헤드를
다음 프레임으로 옮기고 멈추는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 2부터 전역 함수 형태로 지원되었다)

문법 `mc.nextFrame()`

설명

`MovieClip.nextFrame()` 메소드는 전역 함수인 `nextFrame()` 함수와 같은 기능을 가진다. 메소드 형태는 `mc`로 지정된 원격 무비 클립이나 메인 무비를 제어할 때 쓰인다.

일반적인 주의 사항은 `nextFrame()` 전역 함수 부분에 나와있다.

예제

```
// slideshow 클립의 플레이헤드를 한 프레임 앞으로 이동시키고 재생을 멈춘다.  
slideshow.nextFrame();
```

참조

MovieClip.prevFrame(), nextFrame()

MovieClip._parent 속성

호스트 클립 또는 그 클립을 포함하는
무비에 대한 레퍼런스

버전 플래시 4 이후

문법 mc._parent

액세스 읽기 전용

설명

_parent 속성에는 mc를 포함하는 클립 객체에 대한 레퍼런스가 저장된다. 메인 무비는 최상위 레벨에 있기 때문에 _parent 속성이 없다. _root._parent 값은 undefined가 된다. _parent 속성은 현재 클립에 대해 상대적으로 다른 클립을 조작할 때 유용하게 쓰인다.

버그

플래시 5에서는 _parent 속성에 다른 클립을 대입할 수도 있지만 거의 쓸모가 없다. 레퍼런스만 바뀔 뿐 실제 물리적인 구조는 바뀌지 않기 때문이다. 아마 다음 버전에서는 이 부분이 바뀔 것이다.

예제

mc가 무비의 메인 타임라인에 속한다면 mc에서 메인 타임라인을 재생할 때는 다음과 같이 하면 된다.

```
_parent.play();
```

다음과 같이 메인 타임라인의 속성을 바꿀 수도 있다.

```
_parent._alpha = 50;
```

`_parent` 속성을 여러 번 겹쳐서 사용할 수도 있다. 즉 `_parent`의 `_parent`를 액세스할 수도 있다.

```
_parent._parent.play(); // 현재 클립보다 두 단계 위의 클립을 재생한다.
```

참조

`_root`; 13장의 ‘다중 인스턴스 참조하기’

MovieClip.play() 메소드

클립 또는 무비에서 연속적인 프레임 재생을 시작한다.

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 2부터 전역 함수 형태로 지원되었다)

문법 `mc.play()`

설명

`MovieClip.play()` 메소드는 `play()` 전역 함수와 비슷한 기능을 가진다. `mc`로 지정한 원격 무비 클립이나 메인 무비를 제어할 때 메소드 형태를 이용한다.

예제

```
// intro 클립을 재생시킨다.
intro.play();
```

참조

`MovieClip.stop()`, `play()`

MovieClip.prevFrame() 메소드

클립이나 무비의 플레이헤드를 이전 프레임으로 옮기고 멈추는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 2부터 전역 함수 형태로 지원되었다)

문법 `mc.prevFrame()`

설명

`MovieClip.prevFrame()` 메소드는 전역 함수인 `prevFrame()` 함수와 같은 기능을 가진다. 메소드 형태는 `mc`로 지정된 원격 무비 클립이나 메인 무비를 제어할 때 쓰인다.

일반적인 주의 사항은 `prevFrame()` 전역 함수 부분에 나와있다.

예제

```
// slideshow 클립의 플레이헤드를 한 프레임 이전으로 이동시키고 재생을 멈춘다.
slideshow.prevFrame();
```

참조

`MovieClip.nextFrame()`, `prevFrame()`

MovieClip.removeMovieClip() 메소드

플래시 플레이어에서 무비 클립을 삭제하는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 4부터 전역 함수 형태로 지원되었다)

문법 `mc.removeMovieClip()`

설명

`MovieClip.removeMovieClip()` 메소드는 `removeMovieClip()` 전역 함수와 같은 기능을 가진다. `MovieClip` 메소드 형태로 호출하면 `target` 매개변수를 사용하지 않아도 되며 `mc`를 삭제하게 된다. 메소드 형태로 사용하면 오류 발생 위험을 줄일 수 있다.

주의 사항은 `removeMovieClip()` 전역 함수 부분에 나와있다.

참조

`duplicateMovieClip()`, `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`; 13장의 '메소드와 전역 함수가 겹치는 문제'

MovieClip._rotation 속성

클립이나 무비의 회전 각도(도 단위)

버전	플래시 4 이후
문법	<code>mc._rotation</code>
액세스	읽기/쓰기

설명

`_rotation` 속성은 mc를 원래의 방향에서 회전시킨 각도를 나타내는 값이다(mc가 클립이면 원래 방향은 라이브러리에 있는 심벌의 방향을 말한다). 저작 도구에서 변경한 내용과 프로그램을 이용하여 변경한 내용을 모두 `_rotation` 속성에 반영한다. 0 이상 180.0 이하의 값은 클립을 시계 방향으로 회전시키는 것을 의미한다. 0에서 -180.0 사이의 값은 클립을 반시계방향으로 회전시키는 것을 의미한다. 클립을 n 도, $n-360$ 도 회전시키는 것은 결국 똑같다(n 은 양수). 예를 들어 어떤 클립을 299.4도 회전시키는 것과 -60.6도 회전시키는 것은 모두 같은 결과를 만들어낸다. n 이 음수인 경우에도 상관없으므로 n 도와 $n+360$ 도도 똑같다. 예를 들어 클립을 -90도 회전시킨 것은 +270도 회전시킨 것과 같다.

`_rotation`에 -180도와 180도 범위를 벗어나는 값을 집어넣으면 다음과 같은 계산을 거쳐서 그 범위 안에 들어가도록 조절된다.

```
x = newRotation % 360;
if (x > 180) {
    x -= 360;
} else if (x < -180) {
    x += 360;
}
_rotation = x;
```

버그

플래시 4 플레이어에서는 클립의 `_rotation`을 설정하면 클립이 조금씩 작아진다. 따라서 `_rotation` 값을 여러 번 다시 설정하면 클립이 눈에 띄게 작아진다. 이러한 버그를 해결하려면 `_rotation` 값을 설정할 때마다 `_xscale`과 `_yscale`의 값을 100으로 다시 설정해 주어야 한다.

예제

다음과 같은 코드를 클립에 추가하면 프레임을 렌더링할 때마다 클립이 시계 방향으로 5도씩 회전된다.

MovieClip.startDrag() 메소드

무비나 무비 클립이 마우스를 쫓아가도록 만드는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 4부터 전역 함수 형태로 지원되었다)

문법

```
mc.startDrag()
mc.startDrag(lockCenter)
mc.startDrag(lockCenter, left, top, right, bottom)
```

인자

lockCenter mc의 등록 지점이 마우스 포인터의 위치를 따라 움직일지(true) 아니면 원래 위치를 기준으로 움직일지(false) 나타내는 부울 값

left mc의 등록 지점을 드래그할 수 없는 x 좌표의 왼쪽 경계를 나타내는 값

top mc의 등록 지점을 드래그할 수 없는 y 좌표의 위쪽 경계를 나타내는 값

right mc의 등록 지점을 드래그할 수 없는 x 좌표의 오른쪽 경계를 나타내는 값

bottom mc의 등록 지점을 드래그할 수 없는 y 좌표의 아래쪽 경계를 나타내는 값

설명

MovieClip.startDrag() 메소드는 startDrag() 전역 함수와 같은 기능을 하는 메소드이다. MovieClip 형태로 호출하면 target 매개변수가 필요 없이 mc를 드래그한다. 메소드 형태로 사용하면 전역 함수 형태로 사용하는 경우보다 오류 발생 가능성을 줄일 수 있다.

주의 사항은 startDrag() 전역 함수 부분을 참조하기 바란다.

버그

mc를 드래그할 수 있는 영역은 왼쪽, 위, 오른쪽, 아래의 순서대로 입력해야 하지만, 플래시 5 액션스크립트의 Dictionary에서는 그 순서가 잘못 나와있다.

예제

```
// 현재 클립이나 무비를 드래그하여 놓기 위한 버튼 코드
on (press) {
    this.startDrag(true);
}

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

참조

MovieClip.stopDrag(), startDrag(), stopDrag(); 13장의 '메소드와 전역 함수가 겹치는 문제'

MovieClip.stop() 메소드

클립이나 무비의 재생을 중단시키는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 2부터 전역 함수 형태로 지원되었다)

문법 `mc.stop()`

설명

MovieClip.stop() 메소드는 stop() 전역 함수와 같은 역할을 하는 메소드이다. 메소드 형태로 사용하면 mc로 지정한 원격 무비 클립이나 메인 무비를 제어할 수 있다.

일반적인 주의 사항은 stop() 전역 함수를 설명하는 부분에 나와있다.

예제

```
// spinner의 재생을 중단시킴
spinner.stop();
```

참조

MovieClip.play(), stop()

MovieClip.stopDrag() 메소드

진행 중인 드래그 작업을 중단시키는 메소드

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 4부터 전역 함수 형태로 지원되었다)

문법 `mc.stopDrag()`

설명

MovieClip.stopDrag() 메소드는 stopDrag() 전역 함수와 같은 역할을 하는 메소드이다. 하지만 이 메소드를 굳이 사용할 필요는 없다. stopDrag()를 이용하면 무비 클립에서 호출되었든 전역 함수로 호출했든 상관없이 모든 드래그 작업을 취소할 수 있기 때문이다.

참조

MovieClip.startDrag(), stopDrag()

MovieClip.swapDepths() 메소드

인스턴스 스택에 있는 인스턴스의 그래픽 레이어를 변경하는 메소드

버전 플래시 5

문법 `mc.swapDepths(target)`
`mc.swapDepths(depth)`

인자

target mc와 바꿀 무비 클립의 경로를 나타내는 문자열. 무비 클립 레퍼런스를 문자열이 들어갈 자리에 사용하면 경로로 자동 변환되므로 target 자리에 무비 클립 객체 레퍼런스를 사용해도 된다(예: mc.swapDepths(window2)와 mc.swapDepths("window2")).

depth mc의 부모 클립 스택에서의 레벨을 나타내는 정수. depth 값이 -1이면 0보다 밑에, 1이면 0보다 위에, 2이면 2보다 위에 있는 식으로 위치가 결정된다. 자세한 내용은 13장의 '무비와 인스턴스 스택 순서'를 참조하기 바란다. depth 값을 음수로 지정해도 작동하긴 하지만 액션스크립트에서 공식적으로 지원하는 기능은 아니므로, 차기 버전에서 음수 depth를 아예 사용하지 못하게 될 경우를 대비하여 음수는 사용하지 않는 것이 좋다.

설명

.swf 문서에 있는 모든 무비 클립 인스턴스는 플레이어에서 인스턴스의 시각적 레이어를 결정하는 스택에 속하게 된다. 스택은 여러 장의 카드를 쌓아놓은 것과 비슷한 구조를 가진다. 더 높은 위치에 있는 클립이 더 낮은 위치에 있는 클립 위에 있는 것으로 나타난다. 스택에 있는 인스턴스의 위치는 클립이 생성될 때 초기화된다(플레이시 저작 도구 또는 attachMovie()나 duplicateMovieClip()을 이용). swapDepths()를 이용하면 스택에서의 인스턴스 위치를 변경할 수 있다.

swapDepths() 메소드는 두 가지 형식으로 호출할 수 있다. target 인자를 사용하면 mc의 스택 위치를 target과 바꾸는데, 이 때 mc와 target은 같은 부모 클립에 속해야 한다(같은 타임라인에 있어야 함). depth 인자를 사용하면 mc를 mc의 부모 스택에서 새로운 위치로 이동시킨다. 그 위치에 다른 클립이 있으면 원래 있던 클립이 mc가 있던 자리로 움직인다. 해당 부모 클립에 속하지 않는 클립은 swapDepths()를 이용하여 움직일 수 없다. 플레이어에서 무비와 무비 클립의 스택이 어떤 식으로 결정되는지 알고 싶다면, 13장의 '무비와 인스턴스 스택 순서'를 참조하기 바란다.

버그

플레이시 5에서 depth 인자를 이용하여 복사 또는 추가된 인스턴스와 수동으로 만든 인스턴스의 스택 순서를 바꾸면 그림을 다시 그릴 때 문제가 생길 수 있다. depth 인자를 사용할 때는 조심스럽게 코딩해야 하며 depth를 바꾸는 코드는 철저히 테스트하는 습관을 기르자.

참조

13장의 '무비와 인스턴스 스택 순서'

MovieClip._target 속성

클립이나 무비의 대상 경로(슬래시 표기법)

버전 플래시 4 이후**문법** `mc._target`**액세스** 읽기 전용**설명**

`_target` 속성은 플래시 4 형식의 슬래시 표기법으로 mc의 경로를 나타낸 값이다. 예를 들어 ball이라는 클립이 메인 무비 타임라인에 속한다면 ball의 `_target` 속성은 `"/ball"`이 된다. 또한 ball에 속하는 stripe이라는 클립의 `_target` 속성은 `"/ball/stripe"`이 된다.

클립의 경로를 점 표기법으로 나타낼 때는 `targetPath()` 함수를 이용하면 된다.

참조`targetPath()`

MovieClip._totalframes 속성

클립이나 무비 타임라인에 있는 전체 프레임 개수

버전 플래시 4 이후**문법** `mc._totalframes`**액세스** 읽기 전용**설명**

`_totalframes` 속성은 mc(무비 클립 또는 메인 무비)의 타임라인에 있는 프레임 개수를 나타내는 정수이다. 새로 만든 클립이나 메인 무비의 `_totalframes` 값은 언제나 1이다. 하지만 `unloadMovie()`를 이용하여 클립의 내용물을 제거하면 `_totalframes` 값은 0이 된다. 네트워크 속도가 느린 경우 `loadMovie()`를 이용하여 새로운 클립을 로딩하기 직전에 현재 클립을 제거한 상태에서도 순간적으로 0이 될 수 있다. `_totalframes()` 속성은 보통 `_framesloaded` 속성과 함께 프리로더를 만드는 데 주로 쓰인다(MovieClip._framesloaded 부분 참조).

참조

MovieClip._framesloaded

MovieClip.unloadMovie() 메소드

플레이어에서 무비나 무비 클립을 제거한다.

버전 플래시 5부터 메소드 형태로도 쓰임(플래시 3부터 전역 함수 형태로 지원되었다).

문법 `mc.unloadMovie()`

설명

MovieClip.unloadMovie() 메소드는 unloadMovie() 전역 함수와 같은 역할을 하는 메소드이다. MovieClip 메소드 형태로 호출하면 target 매개변수를 사용하지 않아도 되며, mc를 제거한다. 메소드 형태로 사용하면 전역 함수 형태로 사용하는 경우에 비해 오류 발생 가능성이 적다.

주의 사항은 unloadMovie() 전역 함수를 참조하기 바란다.

예제

```
// 1번 레벨에 있는 문서를 제거한다.
_level1.unloadMovie();
```

참조

MovieClip.loadMovie(), unloadMovie(); 13장의 '메소드와 전역 함수가 겹치는 문제'

MovieClip._url 속성

클립이나 무비를 불러온 네트워크 주소

버전 플래시 4 이후

문법 `mc._url`

액세스 읽기 전용

설명

`_url` 속성은 mc의 내용을 불러온 인터넷 또는 로컬 디스크에서의 위치를 나타내는 URL(Uniform Resource Locator) 문자열이다. `_url` 속성은 항상 상대 경로가 아닌 절대 경로 형태로 저장된다. 메인 무비의 `_url` 속성은 현재 .swf 파일의 위치를 나타낸다. .swf 파일에 있는 모든 무비 클립의 `_url` 값은 `MovieClip.loadMovie()`를 통해 각 클립에 외부 .swf 파일을 불러온 경우를 제외하면 메인 무비의 `_url` 속성과 같은 값을 가진다. 외부에서 불러온 .swf 파일을 가지고 있는 클립의 `_url` 속성은 외부에서 불러온 파일의 위치를 가리킨다.

`_url` 속성을 이용하면 원하지 않는 위치에서 무비가 재생되지 않도록 간단한 보안 시스템을 만들 수 있다.

예제

웹사이트에서 불러온 무비의 `_url` 값은 다음과 같은 형식이 된다.

```
"http://www.moock.org/gwen/meetgwen.swf"
```

하드 디스크에 저장되어 있던 파일을 실행시킨 경우에는 `_url` 값이 다음과 같은 형식으로 표현된다.

```
"file:///C:/data/flashfiles/movie.swf"
```

무비가 원하는 위치에 저장되어 있는지 확인하고 싶다면 다음과 같이 하면 된다 (그렇지 않은 경우에는 오류 메시지가 포함된 프레임을 화면에 표시한다).

```
if (_url != "http://www.moock.org/gwen/meetgwen.swf") {  
    trace ("This movie is not running from its intended location.");  
    gotoAndStop("accessDenied");  
}
```

참조

`MovieClip.loadMovie()`

MovieClip.valueOf() 메소드

클립이나 무비의 경로를 문자열로 나타낸 값

버전 플래시 5**문법** `mc.valueOf()`**리턴 값**

점 표기법을 이용하여 mc의 전체 경로를 나타낸 문자열. 예를 들면 다음과 같다.

```

_level11.ball"
_level10"
_level10.shoppingcart.product1"

```

설명

valueOf() 메소드는 점 표기법을 이용하여 mc의 절대 경로를 나타내는 문자열을 리턴한다. MovieClip 객체를 문자열이 들어갈 자리에 집어넣으면 자동으로 valueOf() 메소드가 호출된다. 예를 들어 trace(mc)와 같은 선언문을 사용한 결과는 trace(mc.valueOf())를 실행한 결과와 같다. 따라서 valueOf()를 직접 호출하는 일은 매우 드물다.

참조

Object.valueOf(), targetPath()

MovieClip._visible 속성

클립이나 무비가 화면에 표시되는지 나타내는 값

버전 플래시 4 이후**문법** `mc._visible`**액세스** 읽기/쓰기**설명**

_visible 속성은 mc가 현재 화면에 표시되는지(true) 아닌지(false)를 나타내는 부울 값이다. _visible 속성은 주로 무비를 잠시 화면에 표시되지 않도록 할 때 쓰인다. 숨어있는 클립도 액션스크립트를 이용하여 제어할 수 있으며 재생, 정지, 이벤트 처리와 같은 일반적인 작업을 모두 처리할 수 있다. 숨어있는 클립은 단지 화면에만 표시되지 않을 뿐이다.

완전히 투명하거나 스테이지에서 멀리 떨어져 있는 클립을 포함한 모든 클립의 `_visible` 속성의 초기 값은 `true`이다. `_visible` 속성은 스크립트에서만 변경할 수 있다. 실제 클립이 화면에 출력되는 것에 영향을 미치는 클립의 위치나 투명도와는 상관없이 단순히 클립을 화면에 보이게 하거나 보이지 않게 하는 속성이라고 생각하면 된다.

`_visible`을 `false`로 설정하면 무비 클립을 화면에 표시하기 위해 렌더링하는 시간을 절약할 수 있으므로, 클립을 완전히 투명하게 만들거나 스테이지에서 먼 곳으로 옮겨놓는 것보다는 `_visible` 속성을 이용하여 무비 클립을 숨기는 방법이 더 좋다.

예제

아래 버튼 코드에서는 버튼을 누르면 현재 클립을 숨기고 버튼에서 손가락을 떼면 다시 화면에 클립을 표시한다.

```
on (press) {
    this._visible = false;
}

on (release) {
    this._visible = true;
}
```

참조

`MovieClip._alpha`

MovieClip._width 속성

클립이나 무비의 너비를 픽셀 단위로 나타낸 값

버전	플래시 4(플래시 5에서 개선됨)
문법	<code>mc._width</code>
엑세스	플래시 4에서는 읽기 전용, 플래시 5에서는 읽기/쓰기

설명

`_width` 속성에는 `mc`의 현재 너비를 픽셀 단위로 나타내는 음이 아닌 숫자가 저장된다. `mc`에 아무런 내용도 없다면 `_width` 값은 0이 된다. `_width` 속성은 클립이

나 무비가 차지하는 가장 왼쪽에 있는 픽셀의 위치와 가장 오른쪽에 있는 픽셀의 위치 사이의 거리를 나타낸다(중간에 비어있는 공간이 있어도 상관없다). mc가 차지하고 있는 픽셀은 도형, 그래픽, 버튼, 무비 클립 및 기타 요소를 포함하고 있는 픽셀을 의미한다. 저작 도구나 `_xscale`을 이용하여 클립의 너비를 바꾸면 `_width` 값도 변경된다.

`_width` 값을 설정하여 무비 클립의 수평 방향 크기를 바꿀 수 있다. `_width`에 음수를 대입하려면 그 값은 무시된다. 클립의 `_width` 속성을 0으로 설정하면 클립이 없어지지 않고 한 픽셀짜리 수직선이 된다.

메인 무비의 `_width` 속성(즉 `_root._width` 또는 `_leveln._width`)은 저작 도구의 Modify → Movie → Dimensions에서 지정한 스테이지의 너비가 아니라 메인 무비에 있는 내용물의 너비이다. 스테이지의 너비를 나타내는 속성은 따로 정해져 있지 않다. 필요하다면 스테이지의 너비를 별도의 변수로 만들면 된다. 예를 들어 무비의 스테이지의 너비가 550이라면 다음과 같이 변수를 하나 만들면 된다.

```
_root.stageWidth = 550;
```

다음과 같이 하면 이 값을 어떤 클립의 타임라인에서도 사용할 수 있다.

```
Object.prototype.stageWidth = 550;
```

주의 사항

무비 클립의 너비를 설정하면 그에 비례하여 선의 굵기도 바뀌어 Stroke 패널에서 설정한 원래의 굵기와 다른 선이 된다. 하지만 선의 굵기를 Hairline으로 설정하면 무비 클립의 크기를 변경하더라도 선의 굵기는 바뀌지 않는다. 즉 클립의 크기를 바꾸었을 때 선이 굵어지거나 가늘어지는 것을 방지하려면 Hairline으로 설정하면 된다.

예제

```
ball._width = 20;           // ball의 너비를 20으로 설정한다.
ball._width *= 2;           // ball의 너비를 두 배로 만든다.
```

참조

MovieClip._height, MovieClip._xscale

MovieClip._x 속성

클립이나 무비의 수평 위치를 픽셀 단위로 나타낸 값

버전 플래시 4 이후**문법** `mc._x`**엑세스** 읽기/쓰기**설명**

`_x` 속성은 mc의 등록 지점의 수평 위치를 나타내는 값이지만, 다음과 같은 세 가지 좌표계를 기준으로 그 값이 정해진다.

- mc가 메인 타임라인에 포함되면 `_x`는 스테이지의 왼쪽 끝을 기준으로 측정된다. 예를 들어 `_x`가 20이면 mc의 등록 지점이 스테이지의 왼쪽 끝으로부터 20픽셀 오른쪽에 있다는 것을, `_x`가 -20이면 mc의 등록 지점이 스테이지의 왼쪽 끝으로부터 20픽셀 왼쪽에 있다는 것을 의미한다.
- mc가 다른 무비 클립 인스턴스의 타임라인에 속한다면 `_x`는 그 부모 인스턴스의 등록 지점을 기준으로 측정된다. 예를 들어 `_x`가 20이면 mc의 등록 지점이 부모 인스턴스의 등록 지점에서 오른쪽으로 20픽셀 떨어진 곳에, -20이면 왼쪽으로 20픽셀 떨어진 곳에 있다는 것을 의미한다.
- mc가 메인 무비이면 `_x`는 스테이지의 왼쪽 끝을 기준으로 한 .swf 문서 전체의 수평 방향 오프셋이 된다. 예를 들어 `_x`가 200이면 스테이지의 내용 전체가 처음 만들 때의 위치에서 200픽셀 오른쪽으로 옮겨진 것을, -200이면 200픽셀 왼쪽으로 옮겨진 것을 나타낸다.

`_x` 속성은 플래시의 다른 모든 수평 방향 좌표값과 마찬가지로 오른쪽으로 가면 증가하고 왼쪽으로 가면 감소한다. `_x`에 소수점 이하 부분이 있으면 플래시에서 안티앨리어싱으로 처리한다.

mc를 포함하고 있는 인스턴스의 배율이나 회전 각도가 변경되면 그 좌표계도 함께 변환된다. 예를 들어 mc의 부모 클립이 200% 확대되고 시계 방향으로 90도 회전되면, `_x`는 오른쪽이 아니라 아래쪽으로 갈수록 증가하며 `_x` 값이 1 변할 때마다 2픽셀씩 변하게 된다.

인스턴스와 메인 무비 좌표계 사이에서 값을 바꾸는 작업이 꽤 복잡하긴 하지만, MovieClip 객체에 있는 `localToGlobal()`과 `globalToLocal()` 메소드를 이용하면 비교적 손쉽게 좌표계를 변환할 수 있다.

예제

아래 코드를 클립에 추가하면 매번 프레임이 바뀔 때마다 클립이 5픽셀씩 오른쪽으로 움직인다(부모 클립이 변환되지 않은 경우).

```
onClipEvent (enterFrame) {
    _x += 5;
}
```

`_x`와 `_y`를 이용하여 클립의 위치를 결정하는 작업은 시각적인 액션스크립트 프로그래밍에서 매우 기본적인 작업이다. 아래 예제에서는 클립 이벤트 핸들러를 이용하여 클립이 정해진 속도로 마우스를 쫓아가도록 만든다(온라인 코드 참고에 가면 더 다양한 모션 샘플을 구할 수 있다).

```
// 클립이 마우스를 따라가도록 만든다.
onClipEvent(load) {
    this.speed = 10;

    // 클립을 (leaderX, leaderY) 지점으로 움직인다.
    function follow (clip, leaderX, leaderY) {
        // 목표지점에 다다르지 못했을 경우에만 움직인다.
        if (clip._x != leaderX || clip._y != leaderY) {
            // 클립과 목표 지점 사이의 거리를 계산한다.
            var deltaX = clip._x - leaderX;
            var deltaY = clip._y - leaderY;
            var dist = Math.sqrt((deltaX * deltaX) + (deltaY * deltaY));

            // X와 Y 축 방향의 속도를 정한다.
            var moveX = clip.speed * (deltaX / dist);
            var moveY = clip.speed * (deltaY / dist);

            // 클립이 한 번에 목표 지점에 다다를 만큼 빠르다면 목표
            // 지점까지만 이동하고 그렇지 않다면 클립의 속도만큼
            // 움직인다.
            if (clip.speed >= dist) {
                clip._x = leaderX;
                clip._y = leaderY;
            }
        }
    }
}
```

```
        } else {
            clip._x -= moveX;
            clip._y -= moveY;
        }
    }
}

onClipEvent(enterFrame) {
    follow(this, _root._xmouse, _root._ymouse);
}
```

참조

MovieClip.globalToLocal(), MovieClip.localToGlobal(), MovieClip._y

MovieClip._xmouse 속성

마우스 포인터의 수평 방향 위치

버전 플래시 5

문법 *mc*._xmouse

액세스 읽기 전용

설명

_xmouse 속성에는 mc의 좌표계를 기준으로 마우스 포인터 핫스팟의 수평 위치를 나타내는 부동소수점 값이 저장된다. mc가 메인 무비이면 _xmouse의 값은 스테이지의 왼쪽 끝을 기준으로 결정된다. mc가 인스턴스이면 _xmouse의 값은 인스턴스의 등록 지점을 기준으로 측정된다. 언제나 스테이지를 기준으로 측정한 일관된 _xmouse 값을 사용하고 싶다면 _root._xmouse를 이용하면 된다.

예제

클립에 다음과 같은 코드를 추가하면 클립이 스테이지를 기준으로 하는 마우스 포인터의 수평 위치를 따라 움직이도록 만들 수 있다(직선에서 마우스의 위치를 따라 움직인다).

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

참조

MovieClip._ymouse

MovieClip._xscale 속성

클립이나 무비의 너비를 백분율로 나타낸 값

버전 플래시 4 이후**문법** `mc._xscale`**액세스** 읽기/쓰기**설명**

`_xscale` 속성은 mc 너비의 원본의 너비에 대한 비율을 백분율 단위로 표현한 부동소수점 값이다. mc가 인스턴스인 경우에는 '원본 너비'가 라이브러리에 있는 인스턴스의 심벌의 너비로 정해진다. mc가 메인 무비이면 '원본 너비'는 무비를 만들 때의 타임라인의 너비로 결정된다.

mc의 현재 너비가 원본 너비와 같으면 `_xscale`은 100이 된다. `_xscale` 값이 200이면 mc의 너비가 원본 너비의 두 배가 된다. `_xscale`이 50이면 mc의 너비가 원본 너비의 절반이 된다.

`_xscale` 속성을 변경하면 클립의 너비가 등록 지점을 기준으로 변경된다(등록 지점을 기준으로 각각 왼쪽과 오른쪽으로 비율에 맞추어 크기가 바뀐다). 클립의 크기를 한 방향으로만 변경하려면 클립의 모든 내용을 클립의 심벌에서 정의한 등록 지점을 기준으로 한 쪽에만 놓아야 한다(수평 방향의 프리로더 막대를 만들 때 이러한 방법을 이용한다). 클립의 `_xscale` 값이 음수이면 등록 지점을 기준으로 거울에 비친 상과 같이 좌우가 바뀐 모양이 된다(배율은 양수 값을 사용했을 때와 같은 식으로 결정된다). 클립의 크기는 그대로 두고 좌우만 바꿀 때는 클립의 `_xscale`을 -100으로 설정하면 된다.

예제

```
// ball 객체의 너비를 두 배로 증가시킨다(높이는 바뀌지 않음).
ball._xscale *= 2;

// ball의 좌우를 바꾼다.
ball._xscale = -100;
```

참조

MovieClip._yscale

MovieClip._y 속성

클립이나 무비의 수직 위치를 픽셀 단위로 나타낸 값

버전 플래시 4 이후

문법 *mc._y*

액세스 읽기/쓰기

설명

_y 속성은 mc의 등록 지점의 수직 방향 위치를 나타낸다. _y 속성 값은 다음과 같은 세 가지 좌표계로 측정된다.

- mc가 메인 타임라인에 속하면 _y 속성은 스테이지의 맨 위를 기준으로 측정된다. 예를 들어 _y가 20이면 mc의 등록 지점이 스테이지의 위쪽 끝에서 20픽셀 아래에 있는 것을 나타낸다. -20이면 20픽셀 위에 있는 것을 나타낸다.
- mc가 다른 무비 클립 인스턴스의 타임라인에 속하면 _y는 부모 인스턴스의 등록 지점을 기준으로 결정된다. 예를 들어 _y가 20이면 mc의 등록 지점이 부모 인스턴스의 등록 지점보다 20픽셀 아래에, -20이면 20픽셀 위에 있다는 것을 나타낸다.
- mc가 메인 무비이면 _y 속성은 스테이지의 맨 위를 기준으로 전체 .swf 문서의 수직 방향 오프셋을 나타낸다. 예를 들어 _y가 200이면 스테이지 내용이 무비를 만들 때의 위치에서 200픽셀 아래에, -200이면 200픽셀 위에 있다는 것을 나타낸다.

다른 플래시의 수직 방향 좌표와 마찬가지로 _y 속성은 아래쪽으로 갈수록 증가하고 위로 올라가면 감소한다. 일반적인 직각 좌표계와는 반대로 된다. _y 값에 소수점 이하 부분이 있으면 안티앨리어싱으로 처리된다.

mc를 포함하는 인스턴스의 배율이나 회전 각도가 바뀌면 좌표계도 함께 바뀐다. 예를 들어 mc의 부모 클립이 200% 확대되고 시계 방향으로 90도 회전되면, _y 값은 아래쪽이 아니라 왼쪽으로 갈 때 증가하며 _y가 1만큼 바뀔 때 2픽셀씩 바뀌게 된다.

인스턴스와 메인 무비 좌표계 사이에서 값을 바꾸는 작업은 꽤 복잡하긴 하지만, MovieClip 객체에 있는 `localToGlobal()`과 `globalToLocal()` 메소드를 이용하면 비교적 손쉽게 좌표계를 변환할 수 있다.

예제

클립에 다음과 같은 코드를 집어넣으면 매번 프레임이 바뀔 때마다 클립이 5픽셀씩 아래로 움직인다(부모 클립이 변환되지 않은 경우).

```
onClipEvent (enterFrame) {
    _y += 5;
}
```

참조

MovieClip.globalToLocal(), MovieClip.localToGlobal(), MovieClip._x

MovieClip._ymouse 속성

마우스 포인터의 수직 위치

버전	플래시 5
문법	<code>mc._ymouse</code>
액세스	읽기 전용

설명

`_ymouse` 속성은 mc의 좌표계를 기준으로 마우스 포인터 핫스팟의 수직 위치를 나타낸다. mc가 메인 무비이면 `_ymouse`는 스테이지의 위쪽 모서리를 기준으로 결정된다. mc가 인스턴스이면 `_ymouse`의 값은 인스턴스의 등록 지점을 기준으로 측정된다. 항상 스테이지를 기준으로 하는 `_ymouse` 좌표값을 구하고 싶다면 `_root._ymouse`를 이용하면 된다.

예제

다음과 같은 코드를 클립에 추가하면 스테이지를 기준으로 마우스 포인터의 수직 위치를 따라 클립이 움직이도록 할 수 있다(직선에서 위 아래로 움직인다).

```
onClipEvent (enterFrame) {
    _y = _root._ymouse;
}
```


참조

MovieClip._xmouse

MovieClip._yscale 속성

클립이나 무비의 높이를 백분율로 나타낸 값

버전 플래시 4 이후

문법 *mc._yscale*

액세스 읽기/쓰기

설명

_yscale 속성은 mc 높이의 원본 높이에 대한 비율을 백분율 단위로 표현한 부동소수점 값이다. mc가 인스턴스인 경우에는 '원본 높이'가 라이브러리에서 있는 인스턴스의 심벌의 높이로 정해진다. mc가 메인 무비이면 '원본 높이'는 무비를 만들 때의 타임라인의 높이로 결정된다.

mc의 현재 높이가 원본 높이와 같으면 _yscale은 100이 된다. _yscale 값이 200이면 mc의 높이가 원본 높이의 두 배가 된다. _yscale이 50이면 mc의 높이가 원본 높이의 절반이 된다.

_yscale 속성을 변경하면 클립의 너비가 등록 지점을 기준으로 변경된다(등록 지점을 기준으로 각각 위와 아래로 비율에 맞추어 크기가 바뀐다). 클립의 크기를 한 방향으로만 변경하려면 클립의 모든 내용을 클립의 심벌에서 정의한 등록 지점을 기준으로 한 쪽에만 놓아야 한다(수직 방향의 프리로더 막대를 만들 때 이러한 방법을 이용한다). 클립의 _yscale 값이 음수이면 등록 지점을 기준으로 거울에 비친 상과 같이 위 아래가 바뀐 모양이 된다(배율은 양수 값을 사용했을 때와 같은 식으로 결정된다). 클립의 크기는 그대로 두고 위 아래만 바꿀 때는 클립의 _yscale을 -100으로 설정하면 된다.

예제

```
// ball 객체의 높이를 두 배로 증가시킨다(너비는 바뀌지 않음).
ball._yscale *= 2;
// ball의 위아래를 바꾼다.
ball._yscale = -100;
```

참조

MovieClip._xscale

NaN 전역 속성

잘못된 숫자 데이터를 나타내는 상수(Not-a-Number)

버전 플래시 5**문법** NaN**엑세스** 읽기 전용**설명**

NaN은 잘못된 숫자 값을 나타내는 특별한 상수이다(NaN은 “Not-a-Number”의 약자임). 제대로 된 숫자가 아니면 NaN이 된다. 예를 들면 다음과 같다.

```
Math.sqrt(-1);           // 액션스크립트에서 표현할 수 없는 허수
15 - "coffee cup";      // "coffee cup"은 숫자로 변환될 수 없다.
```

계산할 수 없긴 하지만 NaN도 숫자 데이터이다.

```
typeof NaN;              // "number"가 리턴된다.
```

주의 사항

NaN은 소스 코드에서는 거의 쓰이지 않고 오류 조건을 리턴할 때만 주로 쓰인다. NaN끼리 비교해도 두 개의 NaN이 같은 값으로 간주되지는 않기 때문에, isNaN()을 이용하여 NaN 값을 알아내야 한다. NaN은 Number.NaN을 줄여서 쓴 것이다.

참조

isNaN(), Number.NaN; 4장의 ‘숫자 데이터형에 있는 특수 값’

newline 상수

줄바꿈 문자

버전 플래시 4 이후**문법** `newline`**리턴 값** 줄바꿈 문자**설명**

`newline` 상수는 표준 줄바꿈 문자(ASCII 10)를 나타낸다. “\n” 이스케이프 시퀀스와 똑같은 값을 가지며 텍스트 블록에서 강제로 줄을 바꿀 때 사용한다(보통 텍스트 필드 변수에서 화면에 내용을 출력할 때 사용한다).

주의 사항

플래시 4에서는 `newline`이 함수였지만 플래시 5에서는 속성이나 변수와 같은 형태로 쓰인다. 아래 예제에서는 `newline` 뒤에 괄호를 사용하지 않는다.

예제

```
myOutput = "hello" + newline + "world";
```

참조

4장의 ‘이스케이프 시퀀스’

nextFrame() 전역 함수

무비나 무비 클립의 플레이헤드를 다음 프레임으로 옮기고 정지하는 메소드

버전 플래시 2 이후**문법** `nextFrame()`**설명**

`nextFrame()` 함수는 현재 타임라인의 플레이헤드를 한 프레임 앞으로 이동시키고 그 자리에 정지시킨다. ‘현재 타임라인’은 `nextFrame()` 함수를 호출한 타임라인을 의미한다. `nextFrame()` 함수는 `gotoAndStop(_currentFrame + 1)`를 호출하는 것과 같다. 타임라인의 마지막 프레임에서 `nextFrame()`을 호출하면 그 프레임에서

플레이헤드를 정지시킨다. 현재 장면 뒤에 다음 장면이 있으면 `nextFrame()`에서 플레이헤드를 다음 장면의 1번 프레임으로 이동시킨다.

참조

`gotoAndStop()`, `MovieClip.nextFrame()`, `nextScene()`, `prevFrame()`

nextScene() 전역 함수

무비의 플레이헤드를 다음 장면의 1번 프레임으로 이동시키는 메소드

버전 플래시 2 이후

문법 `nextScene()`

설명

`nextScene()` 함수는 무비의 메인 플레이헤드를 다음 장면의 1번 프레임으로 옮기고 메인 무비 타임라인을 정지시킨다. '현재 장면'은 `nextScene()` 함수를 호출한 장면을 의미한다. 이 함수는 메인 타임라인에서 호출해야 한다. 즉 무비 클립이나 `onClipEvent()` 핸들러에서는 `nextScene()`이 제대로 작동하지 않는다. 무비의 마지막 장면에서 이 함수를 호출하면 플레이헤드를 현재 장면의 1번 프레임으로 움직이고 무비를 정지시킨다.

참조

`nextFrame()`, `prevScene()`

Number() 전역 함수

주어진 값을 숫자 데이터형으로 변환하는 메소드

버전 플래시 5

문법 `Number(value)`

인자

value 숫자로 변환할 값을 나타내는 표현식

리턴 값

value를 숫자형으로 변환한 값

설명

Number() 함수는 주어진 인자를 숫자형으로 바꾸고 그 결과를 리턴한다. 다양한 데이터형을 숫자로 변환하는 방법은 [표 3-1]에 나와 있다. 대부분의 경우에 Number() 함수를 직접 사용할 필요는 없다. 액션스크립트에서 필요에 따라 자동으로 값을 변환해주기 때문이다.

Number() 함수와 Number 클래스 생성자를 혼동하지 않도록 주의하자. Number 함수는 주어진 숫자를 숫자형으로 변환하는 함수이고 Number 클래스는 원시 숫자 데이터를 속성이나 메소드를 가질 수 있는 객체로 감싸는 데 쓰이는 클래스이다.

주의 사항

Number() 함수는 플래시 5 형식으로 변환된 플래시 4의 .fla 파일에서 혼하게 볼 수 있다. 플래시 4 파일을 플래시 5 형식으로 변환할 때 데이터형을 처리하는 방법에 대한 내용은 3장의 '플래시 4에서 플래시 5로의 데이터형 변환'에서 찾을 수 있다.

참조

Number 클래스, parseFloat(), parseInt(); 3장의 '직접 형 변환'

Number 클래스

숫자 데이터를 위한 래퍼 클래스

버전 플래시 5

생성자 new Number(value)

인자

value 그 값을 계산하고 필요에 따라 숫자 값으로 변환한 후 Number 객체로 감쌀 표현식

클래스 속성

아래 속성은 Number.propertyName과 같이 Number 클래스의 속성으로 직접 액세스할 수 있다. 이 속성들을 액세스하기 위해 Number 객체의 인스턴스를 별도로 만들지 않아도 된다(즉 생성자 함수를 사용할 필요가 없다). NaN과 같은 몇몇 속성을 액세스할 때는 Number.propertyName 표기법을 쓰지 않아도 된다. Number.NaN 대신 NaN만 적어도 된다(각 속성별로 자세한 내용은 뒤에서 따로 설명하겠다).

MAX_VALUE

액션스크립트에서 표현할 수 있는 가장 큰 양수

MIN_VALUE

액션스크립트에서 표현할 수 있는 가장 작은 양수

NaN

잘못된 숫자 데이터를 표현하기 위한 숫자가 아닌 값(Not-a-Number)

NEGATIVE_INFINITY

-MAX_VALUE보다 작은 수

POSITIVE_INFINITY

MAX_VALUE보다 큰 수

메소드

toString() 숫자를 문자열로 변환하는 메소드

설명

Number 클래스는 두 가지 용도로 쓰인다.

- 숫자 데이터가 유효한지 확인할 때 쓸 수 있는 특별한 숫자 값을 나타내는 내장 속성(MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY)을 액세스할 때
- 10진수와 16진수와 같이 서로 다른 수 체계끼리 값을 변환할 때. Number.toString() 메소드 참조

주의 사항

클래스 속성을 액세스할 때는 Number 객체를 새로 만들지 않아도 된다. 가능하다면 그 클래스 속성에 해당하는 전역 속성(NaN, Infinity, -Infinity)을 사용하는 것이 더 편하다. 사실 Number 클래스 속성을 따로 써야 하는 경우는 거의 없다.

하지만 Number 클래스의 toString() 메소드를 사용할 때는 Number 객체를 만들어야 한다. 하지만 원시 숫자 값에 대해 메소드를 호출하면 인터프리터에서 자동으로 Number 객체를 만들어준다.

```
x = 102;
x.toString(16); // 이 연산을 처리하기 전에 x는 자동으로
                // Number 객체로 변환된다.
```

toString()을 이용하여 여러 가지 진법으로 값을 변환할 때는 이러한 방법을 많이 사용한다. 하지만 이러한 작업은 그다지 자주 처리하지 않으므로 Number 클래스를 사용할 일은 그리 많지 않다.

참조

Math 객체; 4장의 '숫자 유형'

Number.MAX_VALUE 속성

액션스크립트에서 표현할 수 있는 가장 큰 양수

버전	플래시 5
문법	Number.MAX_VALUE
액세스	읽기 전용

설명

MAX_VALUE 속성에는 액션스크립트에서 표현할 수 있는 가장 큰 양수(1.7976 931348623157e+308)가 저장되어 있다. 아래 예제에 나온 것처럼 매우 큰 숫자가 필요한 경우에 유용하다. 액션스크립트에서 MAX_VALUE보다 큰 값은 표현할 수 없기 때문에 POSITIVE_INFINITY로 간주된다.

예제

아래 예제는 배열에 있는 가장 작은 원소를 찾는 코드이다. minVal 변수를 MAX_VALUE로 초기화하여 어떤 값이 나오더라도 초기 값보다 작은 값이 될 수 있도록 하였다.

```
var myArray = [-10, 12, 99]
var minVal = Number.MAX_VALUE
for (thisValue in myArray) {
    if (myArray[thisValue] < minVal) {
        minVal = myArray[thisValue];
    }
}
trace ("The minimum value is " + minVal);
```

참조

Number.MIN_VALUE, Number.POSITIVE_INFINITY; 4장의 '숫자 데이터형에 있는 특수 값'

Number.MIN_VALUE 속성

액션 스크립트에서 표현할 수 있는 가장 작은 양수

버전 플래시 5

문법 Number.MIN_VALUE

액세스 읽기 전용

설명

MIN_VALUE 속성에는 액션스크립트에서 사용할 수 있는 가장 작은 양수인 5e-324가 저장된다(액션스크립트에서 표현할 수 있는 가장 작은 음수인 -Number.MAX_VALUE와 혼동하지 않도록 주의하자).

참조

Number.MAX_VALUE; 4장의 '숫자 데이터형에 있는 특수 값'

Number.NaN 속성

잘못된 숫자 데이터를 나타내는 상수(Not-a-Number)

버전	플래시 5
문법	Number.NaN
액세스	읽기 전용

설명

NaN 속성에는 숫자 데이터형의 '잘못된 숫자' 값이 저장된다. 논리적으로 불가능한 수학 계산 결과(예: 0/0)를 표현하거나 숫자로 표현할 수 없는 데이터 변환을 나타낼 때 이 값을 사용한다.

보통 Number.NaN보다는 전역 속성인 NaN을 더 많이 사용한다.

주의 사항

NaN을 소스 코드에서 사용하는 경우는 거의 없고 보통 어떤 연산을 처리하는 경우 오류 조건을 리턴할 때 쓰인다. NaN을 확인하려면 전역 함수인 isNaN()을 사용해야 한다.

참조

isNaN(), NaN; 4장의 '숫자 데이터형에 있는 특수 값'

Number.NEGATIVE_INFINITY 속성

-Number.MAX_VALUE 보다 작은 값을 나타내는 상수

버전	플래시 5
문법	Number.NEGATIVE_INFINITY
액세스	읽기 전용

설명

NEGATIVE_INFINITY 속성에는 -Number.MAX_VALUE(액션스크립트에서 사용할 수 있는 가장 작은 음수)보다 작은 값을 나타내는 특별한 숫자 값이 저장된다. '언더플로우 조건(underflow condition)' 이라고도 부르며 이 값은 보통 수학적인 오류 때문에 생긴다.

보통 `Number.NEGATIVE_INFINITY`보다는 전역 속성인 `-Infinity`를 더 많이 사용한다.

참조

`-Infinity`, `isFinite()`, `Number.MAX_VALUE`; 4장의 '숫자 데이터형에 있는 특수 값'

Number.POSITIVE_INFINITY

`Number.MAX_VALUE`보다 큰 값을 나타내는 상수

버전 플래시 5

문법 `Number.POSITIVE_INFINITY`

액세스 읽기 전용

설명

`POSITIVE_INFINITY` 속성에는 `Number.MAX_VALUE`(액션스크립트에서 사용할 수 있는 가장 큰 수)보다 더 큰 값을 나타내기 위한 특별한 숫자 값이 저장된다. 오버플로우 조건이라고도 부르며 이 값은 보통 수학적 오류 때문에 생긴다.

보통 `Number.POSITIVE_INFINITY`보다는 전역 속성인 `Infinity`를 더 많이 사용한다.

예제

```
if (score == Number.POSITIVE_INFINITY) {
    trace ("You've achieved the highest possible score.");
}
```

참조

`Infinity`, `isFinite()`, `Number.MAX_VALUE`; 4장의 '숫자 데이터형에 있는 특수 값'

Number.toString() 메소드

Number를 문자열로 변환하는 메소드

버전 플래시 5

문법 `numberObject.toString(radix)`

인자

radix numberObject를 문자열 형식으로 표현할 때 사용할 진법의 밑을 나타내는 2 이상 36 이하의 정수. 이 인자를 생략하면 10이 기본값으로 쓰인다(옵션).

리턴 값

numberObject를 문자열로 변환한 값

설명

`toString()` 메소드에서는 Number 객체의 값을 구해서 그 값을 문자열로 변환하여 그 문자열을 리턴한다. radix 인자를 이용하면 여러 가지 진법으로 숫자를 변환할 수 있다(예: 2진법, 8진법, 10진법, 16진법 등). 10에서 35까지의 숫자는 각각 A부터 Z까지의 글자로 표현한다. 일반적으로는 (16진수에서 10에서 15까지를 표현하기 위해) A부터 F까지만을 사용한다.

`Number.toString()`을 원시 숫자 값에 사용하려면 다음과 같이 괄호로 그 값을 감싸면 된다.

```
(204).toString(16);
```

예제

```
x = new Number(255);
trace(x.toString());           // "255"가 출력된다(10진법).
trace(x.toString(16));         // "ff"가 출력된다(16진법).
trace(x.toString(2));          // "11111111"이 출력된다(2진법).

// 16진수 리터럴을 10진수로 변환한다.
trace((0xFFFFF).toString(10)); // "16777215"가 출력된다.
```

참조

`Number()`, `Object.toString()`, `parseInt()`; 4장의 '정수 리터럴'

Object 클래스

다른 모든 클래스와 범용 객체의 기반이 되는 클래스

버전 플래시 5

생성자 `new Object()`

속성

constructor 객체를 만들기 위한 클래스 생성자 함수에 대한 레퍼런스

__proto__ 객체의 생성자 함수의 prototype 속성에 대한 레퍼런스

메소드

toString() 객체 값을 문자열로 변환하는 메소드

valueOf() 객체의 원시 값이 있다면 그 값을 리턴하는 메소드

설명

Object 클래스는 액션스크립트 객체 모델의 기반이 되는 클래스이다. Object는 두 가지 일반적인 용도로 쓰인다. 첫째 새로운 범용 객체를 만들기 위한 생성자이며, 둘째 새로운 클래스의 기반이 되는 슈퍼클래스이다. 액션스크립트에 있는 모든 클래스는 사용자 정의 클래스와 내장 클래스를 불문하고 모두 Object 클래스로부터 파생된다. 따라서 모든 클래스의 모든 객체에서 Object 클래스의 속성을 상속받는다(클래스에 따라 그러한 속성을 무효화하는 것도 있다).

생성자를 사용하지 않고 코드에서 Object 클래스의 범용 객체를 직접 만들려면 문자열 리터럴이나 배열 리터럴을 사용하는 것처럼 객체 리터럴을 사용하면 된다. 객체 리터럴은 속성의 이름과 값의 쌍을 쉼표로 구분하여 열거하고 전체를 중괄호로 감싼 형태로 만든다. 일반적인 문법은 다음과 같다.

```
{ property1: value1, property2: value2, property3: value3 }
```

객체 리터럴에 있는 속성의 이름은 '14장. 렉시컬 구조'에서 설명한 규칙에 맞추어 만들어야 한다. 속성 값에는 어떤 표현식을 사용해도 된다.

```
// 두 개의 숫자 속성을 가지는 객체 리터럴
myObject = { x: 30, y: 23 };
// 복합 표현식을 이용하여 x 속성을 설정해도 된다.
myOtherObject = { x: Math.floor(Math.random() * 50 + 1) };
```

객체 리터럴은 언제나 범용 익명 객체가 되므로 객체 형식의 데이터를 임시로 사용해야 하는 경우에만 객체 리터럴을 사용한다(예: `Sound.setTransform()`, `Color.setTransform()`, `MovieClip.localToGlobal()`과 같은 메소드를 호출하는 경우).

참조

‘12장. 객체와 클래스’

Object.constructor 속성

객체를 만들기 위한 클래스 생성자에 대한 레퍼런스

버전 플래시 5

문법 `someObject.constructor`

액세스 읽기/쓰기(객체의 `constructor` 속성에 다른 값을 대입하면 클래스 상속 구조가 바뀌기 때문에 되도록이면 그렇게 하지 않는 것이 좋다)

설명

`constructor` 속성에는 `someObject`를 만들기 위해 사용한 생성자 함수에 대한 레퍼런스가 저장된다. 예를 들어 `Date` 객체의 `constructor` 속성은 `Date` 생성자 함수가 된다.

```
now = new Date();
now.constructor == Date;           // true
```

참조

12장의 ‘constructor 속성’

Object.__proto__ 속성

객체의 생성자의 `prototype` 속성에 대한 레퍼런스

버전 플래시 5

문법 `someObject.__proto__`

액세스 읽기/쓰기(객체의 `__proto__` 속성 값을 바꾸면 클래스 상속 구조가 변경되므로 되도록이면 그렇게 하지 않는 것이 좋다).

설명

`__proto__` 속성에는 속성을 클래스 계층을 따라 아래로 전달하는 데 쓰이는 `someObject` 생성자 함수의 `prototype` 속성(자동으로 대입됨)에 대한 레퍼런스가 저장된다. 객체의 `__proto__` 속성은 어떤 객체에서 상속된 속성을 찾아보기 위해 인터프리터에서 내부적으로 쓰이는 것이 대부분이지만 [예제 12-6]이나 [예제 12-9]에 나온 것처럼 이 속성을 이용하여 객체가 속하는 클래스를 알아낼 수도 있다. `__proto__` 속성의 앞과 뒤에는 밑줄이 각각 두 개씩 있다는 것에 주의하자.

참조

12장의 '`__proto__` 속성'

Object.toString() 메소드

객체 값을 문자열로 리턴하는 메소드

버전 플래시 5

문법 `someObject.toString()`

리턴 값

객체를 기숀 또는 표현할 수 있는 내부적으로 정의된 문자열

설명

`toString()` 메소드는 `someObject`를 기숀하는 문자열을 리턴한다. 기본적으로 `someObject.toString()`에서는 다음과 같은 표현식을 리턴한다.

```
"[object " + class + "]"
```

여기서 `class`는 `someObject`가 속하는 클래스의 내부적으로 정의된 이름이다. 문자열이 들어갈 자리에 `someObject`를 사용하면 액션스크립트 인터프리터에서 자동으로 `toString()` 메소드를 호출한다.

```
x = new Object();
trace(x);          // "[object Object]"가 출력된다.
```

대부분의 클래스에서는 `Object`의 `toString()` 메소드 대신 새로운 함수를 정의하여 클래스의 각 멤버에 대한 유용한 정보를 제공한다. 예를 들어 `Date.toString()` 메

소드에서는 날짜와 시각을, `Array.toString()` 메소드에서는 배열의 각 원소를 쉼표로 구분한 문자열을 리턴한다. 직접 클래스를 만들 때도 `toString()` 메소드를 오버라이드하여 유용한 정보를 제공할 수 있다.

예제

이 예제에서는 12장에서 만든 `Ball` 클래스에서 사용자 정의 `toString()` 메소드를 제공하는 법을 보여준다.

```
// 사용자 정의 toString() 메소드를 추가하는 법
// Ball 생성자를 만든다.
function Ball (radius, color, xPosition, yPosition) {
    this.radius = radius;
    this.color = color;
    this.xPosition = xPosition;
    this.yPosition = yPosition;
}

// Ball 클래스의 prototype의 toString() 메소드에 함수 리터럴을 대입한다.
Ball.prototype.toString = function () {
    return "A ball with the radius " + this.radius;
};

// 새로운 ball 객체를 만든다.
myBall = new Ball(6, 0x00FF00, 145, 200);

// myBall의 문자열 값을 확인한다.
trace(myBall); // "A ball with the radius 6"이 출력된다.
```

참조

`Array.toString()`, `Date.toString()`, `Number.toString()`, `Object.valueOf()`

Object.valueOf() 메소드

객체의 원시값을 리턴하는 메소드

버전 플래시 5

문법 `someObject.valueOf()`

리턴 값

someObject와 어떤 원시값이 연결되어 있으면 someObject에서 내부적으로 정의된 원시값을 리턴한다. someObject에 연결된 원시값이 없으면 someObject 자체가 리턴된다.

설명

valueOf() 메소드는 객체와 연관된 원시값이 있다면 그 데이터를 리턴한다. 이 메소드는 보통 원시 데이터와 연관된 Number, String, Boolean 클래스에서 주로 사용한다. MovieClip.valueOf() 메소드에서는 클립에 대한 경로를 나타내는 문자열을 리턴한다.

valueOf()를 직접 호출하는 일은 거의 없다. 원시값이 들어갈 자리에 someObject를 사용하면 인터프리터에서 자동으로 이 메소드를 호출한다.

참조

Boolean.valueOf(), MovieClip.valueOf(), Number.valueOf(), Object.toString(), String.valueOf()

parseFloat() 전역 함수

문자열에서 부동소수점 소수를 추출하는 함수

버전 플래시 5

문법 parseFloat(stringExpression)

인자

stringExpression 부동소수점 소수를 추출할 문자열

리턴 값

추출된 부동소수점 소수. 추출하지 못하는 경우에는 NaN을 리턴한다.

설명

parseFloat() 함수는 문자열을 부동소수점 소수(소수 부분이 있는 숫자)로 변환하는 함수이다. 부동소수점 소수를 문자열 형식으로 나타내는 표현식에서만 작동하며 그렇지 않다면 NaN을 리턴한다. 문자열은 다음과 같은 형식으로 구성해야 한다.

- 앞에 공백이 들어갈 수 있다.
- +나 -와 같이 부호를 나타내는 문자가 들어갈 수 있다.
- 적어도 하나의 0 이상 9 이하의 숫자가 들어가고 소수점이 들어갈 수 있다.
- e나 E로 시작하고 뒤에 정수가 붙는 지수 표현을 사용할 수 있다.

문자열의 뒷부분에 앞에 있는 숫자와 연결되지 않은 문자들이 있으면 그 문자들은 무시된다.

주의 사항

사용자가 텍스트 필드에 입력한 데이터는 언제나 string 데이터형이기 때문에, `parseFloat()`을 이용하여 사용자가 입력한 텍스트를 숫자로 변환해야 한다. `parseFloat()`에서는 숫자와 숫자가 아닌 문자를 포함한 문자열에서 부동소수점 소수를 추출할 수 있지만 `Number()`에서는 그렇게 할 수 없다는 점에 주의하자.

예제

```
parseFloat("14.5 apples");      // 14.5
parseFloat(".123");              // 0.123
var x = "15, 4, 23, 9";
parseFloat(x);                  // 15
```

참조

`isNaN()`, `NaN`, `Number()`, `parseInt()`; 3장의 '직접 형 변환', 4장의 '정수와 부동소수점수'

`parseInt()` 전역 함수

문자열에서 정수를 추출하거나 숫자를
10진수로 변환하는 함수

버전 플래시 5

문법 `parseInt(stringExpression)`
 `parseInt(stringExpression, radix)`

인자***stringExpression***

정수를 추출할 문자열

radix

추출할 정수의 밑(진법)을 나타내는 2 이상 36 이하의 정수. 이 값을 지정하지 않으면 *stringExpression*의 내용에 따라 진법이 결정된다 (아래 내용 참조).

리턴 값

추출된 정수 값(원래의 *radix*와는 상관없음). 추출에 성공하지 못하면 NaN을 리턴한다.

설명

`parseInt()` 함수는 문자열 표현식을 정수로 변환한다. 주어진 진법의 정수를 표현한 문자열에서만 제대로 작동한다. 문자열은 다음과 같은 형식으로 구성되어야 한다.

- 앞에 공백이 들어갈 수 있다.
- +나 -와 같이 부호를 나타내는 문자가 들어갈 수 있다.
- 적어도 하나의 0 이상 9 이하의 숫자가 들어가고 소수점이 들어갈 수 있다.

숫자 형태로 추출할 수 없는 문자들이 뒤에 덧붙어 있으면 그러한 문자들은 무시된다.

이 함수에서 추출한 숫자는 주어진 문자열에서 처음으로 나오는 공백이 아닌 문자로 시작해서 주어진 *radix*에 속하지 않는 첫 번째 문자 앞에 있는 숫자로 끝난다. 예를 들어 10진법에서는 F를 사용할 수 없기 때문에 다음과 같이 함수를 호출하면 2가 리턴된다.

```
parseInt("2F", 10);
```

하지만 16진법에서는 A, B, C, D, E, F까지 모두 숫자로 간주되므로 다음과 같이 함수를 호출하면 47이 리턴된다.

```
parseInt("2F", 16); // 16진수 2F를 10진수로 변환하면 47이 된다.
```

parseInt()의 radix 인자는 문자열에 있는 숫자의 진법을 나타낸다. 즉 radix 인자를 이용하면 인터프리터에 '이 숫자를 16진수로 간주하라', '이 숫자를 2진법으로 처리해라'와 같은 명령을 내릴 수 있다.

parseInt()에서는 0x로 시작하는 문자열은 16진수로(radix에 16을 전달한 것과 똑같은), 0으로 시작하는 수는 8진수로(radix에 8을 전달한 것과 같은) 간주한다.

```
parseInt("0xFF");    // 16진수로 간주되므로 255가 된다.
parseInt("FF", 16);   // 16진수로 간주되므로 255가 된다.
parseInt("0377");     // 8진수로 간주되므로 255 = (3 * 64) + (7 * 8) +
                      // (7 * 1) 가 된다.
parseInt("377", 8);    // 8진수로 간주되므로 255가 된다.
```

하지만 radix 값을 직접 지정해 주면 그러한 규칙이 적용되지 않는다.

```
parseInt("0xFF", 10)   // 10진수로 간주되므로 0이 된다.
parseInt("0x15", 10)   // 10진수로 간주되므로 0이 된다(15도, 21도 아니다).
parseInt("0377", 10)   // 10진수로 간주되어 377이 된다.
```

parseInt() 함수에서는 10진수만 사용할 수 있지만, 소수 부분도 추출할 수 있는 parseFloat()과는 달리 정수만을 추출할 수 있다는 점에 주의하자.

예제

parseInt()는 주로 숫자와 텍스트를 모두 포함하는 문자열에서 정수를 추출하거나 어떤 숫자의 소수점 이하 부분을 잘라낼 때(Math.floor())와 유사함) 쓰인다.

```
parseInt("Wow, 20 people were there"); // NaN
parseInt("20 people were there");       // 20
parseInt("1001", 2);                     // 9 (1001을 2진수로 간주한 경우)
parseInt(1.5); // 1 (숫자 1.5를 문자열로 변환한 다음 parseInt() 함수가 적용됨)
```

참조

Math.floor(), NaN, parseFloat(): 3장의 '직접 형 변환', 4장의 '정수와 부동소수점수'

play() 전역 함수

무비에서 프레임을 순서대로 화면에 표시하는 작업을 시작하는 함수

버전 플래시 2 이후

문법 `play()`

설명

`play()` 함수를 호출하면 현재 메인 무비 또는 무비 클립에 있는 프레임을 순서대로 화면에 표시하는 작업이 시작된다. 현재 무비 또는 무비 클립이란 `play()` 함수를 호출하는 선언문이 포함된 무비 또는 무비 클립을 의미한다. 프레임은 Movie Property(Modify → Movie → Frame Rate)에서 설정한 전체 무비의 초당 프레임 수(FPS, frames per second) 설정에 따라 화면에 표시된다.

일단 무비가 시작되면 무비나 무비 클립은 재생을 중단시키는 다른 함수를 호출하기 전까지 계속해서 재생된다. 모든 무비 클립은 플레이헤드가 타임라인의 끝에 도착하면 다시 첫 번째 프레임부터 재생된다. 하지만 브라우저에서는 무비를 HTML 페이지에 임베드하는 데 사용한 코드에서 LOOP 속성을 이용하여 무비가 계속 루프를 돌도록 지정한 경우에만 무비가 다시 첫 번째 프레임부터 시작된다(Publish 명령을 이용하여 HTML 페이지에 무비를 임베드한다면 File → Publish Settings → HTML → Playback → Loop를 선택하여 LOOP 속성을 설정할 수 있다).

참조

`gotoAndPlay()`, `MovieClip.play()`, `stop()`

prevFrame() 전역 함수

무비의 플레이헤드를 이전 프레임으로 이동시키고 멈추는 함수

버전 플래시 2 이후

문법 `prevFrame()`

설명

`prevFrame()` 함수는 현재 타임라인의 플레이헤드를 이전 프레임으로 이동시키고 거기에서 멈춘다. '현재 타임라인'은 `prevFrame()`을 호출한 타임라인을 뜻한다.

다. `prevFrame()` 함수는 `gotoAndStop(_currentFrame - 1)`과 똑같다. 타임라인의 첫 번째 프레임에서 이 함수를 호출하면 그 프레임에서 플레이헤드를 멈추기만 한다. 하지만 현재 장면 앞에 다른 장면이 있다면 플레이헤드를 앞 장면의 마지막 프레임으로 이동시킨다.

참조

`gotoAndPlay()`, `MovieClip.prevFrame()`, `nextFrame()`, `prevScene()`

prevScene() 전역 함수

무비의 플레이헤드를 앞 장면의 첫 번째 프레임으로 이동시키는 함수

버전 플래시 2 이후

문법 `prevScene()`

설명

`prevScene()` 함수는 무비의 메인 플레이헤드를 현재 장면 앞 장면의 첫 번째 프레임으로 이동시키고 나서 메인 무비 타임라인을 정지시킨다. ‘현재 장면’은 `prevScene()`을 호출한 장면을 뜻한다. 이 함수는 어떤 장면의 메인 타임라인에서 호출해야만 제대로 작동한다. 즉 무비 클립이나 `onClipEvent()` 핸들러에서 이 함수를 호출하면 이 함수가 작동하지 않는다. 무비의 첫 번째 장면에서 `prevScene()`을 호출하면 플레이헤드를 그 장면의 첫 번째 프레임으로 이동시킨 후 무비 재생을 멈춘다.

참조

`nextScene()`, `prevFrame()`

print() 전역 함수

무비나 무비 클립의 프레임을 벡터 형식으로 인쇄하는 함수

버전 플래시 5

문법 `print(target, boundingBox)`

인자

target 인쇄할 무비 클립이나 문서 레벨의 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 변환된다)

boundingBox

target을 인쇄할 때 프레임을 자르는 방법을 나타내는 문자열. 인쇄할 모든 페이지를 나타내는 경계 상자에 의해 화면을 잘라낸다. 인쇄할 페이지에 포함될 target의 영역은 boundingBox(반드시 문자열 리터럴로 지정해야 한다)에서 다음과 같은 세 가지 값 중 하나로 지정할 수 있다.

“bframe”

각 출력 프레임의 경계 상자는 각 프레임의 내용을 모두 인쇄할 수 있을 만한 크기로 각각 설정된다. 각 프레임의 배율을 조절하여 화면을 가득 채울 수 있도록 만든다.

“bmax”

모든 프레임의 내용이 차지하는 영역을 조합하여 일반적인 경계 상자를 만든다. 각 프레임의 배율과 위치는 일반적인 경계 상자에 대한 상대적인 비율에 맞게 조절된다.

“bmovie”

target 클립에 있는 프레임을 하나 지정하여 모든 인쇄할 프레임의 경계 상자 크기를 그 프레임에 맞춘다. 인쇄할 프레임의 내용은 지정된 프레임의 경계 상자에 맞추어 잘라낸다. 프레임의 레이블을 #b로 지정해 두면 그 프레임이 경계 상자가 된다.

설명

웹 브라우저의 내장 인쇄 기능을 이용하여 플래시 무비를 인쇄하면 일관된 인쇄 결과를 얻기도 힘들고 화질이 떨어질 수도 있다. print() 함수를 이용하면 플래시에서 정확하게, 그리고 좋은 화질로 무비의 내용을 인쇄할 수 있다. 기본적으로 print()에서는 target의 타임라인에 있는 모든 프레임을 한 페이지에 한 프레임씩, 그리고 boundingBox 인자로 지정한 대로 잘라서 프린터로 인쇄한다. 특정 프레임만 인쇄하고 싶다면 인쇄할 프레임의 레이블을 #P로 지정하면 된다.

print() 함수에서는 포스트스크립트 프린터에는 벡터를 직접 전송하고 포스트스크립트를 지원하지 않는 프린터에는 벡터를 비트맵으로 변환하여 전송한다.

print()에서는 벡터를 사용하기 때문에 알파값이나 색을 변환한 무비는 인쇄할 수 없다. 색 효과가 들어있는 무비를 인쇄하려면 printAsBitmap() 함수를 이용해야 한다.

주의 사항

플래시 4r20 이상에서는 getURL() 액션을 약간 수정하여 다양한 플래시 5 print() 함수 기능을 구현할 수 있다. 자세한 내용은 아래 사이트에 있는 매크로미디어 플래시 프린팅 SDK를 참조하기 바란다.

<http://www.macromedia.com/software/flash/open/webprinting/authoring.html>

예제

```
// 메인 무비 타임라인에 있는 모든 프레임을 인쇄한다.  
// 한 페이지에 한 프레임씩 인쇄한다.  
print("_root", "bframe");  
  
// 메인 무비 타임라인에 있는 모든 프레임을 인쇄한다.  
// 각 프레임의 크기는 모든 프레임을 조합한 크기에 대해 상대적으로 조절된다.  
print("_root", "bmax");
```

다음과 같은 코드가 들어있는 버튼을 클릭하면 플래시에서 버튼의 타임라인에 있는 모든 프레임을 인쇄한다. 이 때 #b라는 레이블이 붙어 있는 프레임의 경계 상자를 기준으로 모든 프레임을 잘라내며 각 프레임이 한 장에 꼭 차도록 배율을 조절한다.

```
on (release) {  
    print(this, "bmovie");  
}
```

참조

getURL(), printAsBitmap(), printAsBitmapNum(), printNum()

printAsBitmap() 전역 함수

무비나 무비 클립의 프레임을 비트맵으로 인쇄하는 함수

버전 플래시 5**문법** `printAsBitmap(target, boundingBox)`**인자****target** 인쇄할 무비 클립이나 문서 레벨의 경로를 나타내는 문자열 또는 레퍼런스 (문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 자동 변환된다)**boundingBox**`print()`에서 설명한 것과 같이 `target`의 프레임을 인쇄할 때 프레임을 자르는 방법을 나타내는 문자열**설명**

`printAsBitmap()` 함수는 기능적으로 `print()`와 같지만 프린터에 데이터를 전송할 때 벡터가 아닌 래스터 그래픽으로 인쇄한다. 따라서 색 변환이 포함된 무비도 제대로 인쇄할 수 있지만 벡터를 기반으로 만든 그래픽을 인쇄하면 화질이 조금 떨어진다.

주의 사항`print()` 함수에 있는 주의 사항 참조**참조**`print()`, `printAsBitmapNum()`, `printNum()`

printAsBitmapNum() 전역 함수

특정 문서 레벨에 있는 프레임을 비트맵으로 인쇄한다.

버전 플래시 5**문법** `printAsBitmapNum(level, boundingBox)`

인자

level 인쇄할 문서 레벨을 나타내는 음이 아닌 정수 또는 음이 아닌 정수 값을 가지는 표현식

boundingBox

print()에서 설명한 것과 같이 level에 있는 프레임을 인쇄할 때 프레임의 자르는 방법을 나타내는 문자열

설명

printAsBitmapNum() 함수는 printAsBitmap() 함수와 거의 똑같지만 인쇄할 레벨을 문자열이 아니라 숫자로 지정한다는 점에서 차이가 난다. 즉 printAsBitmapNum()에서는 무비 클립이 아니라 문서 레벨만을 인쇄할 수 있다. 보통 인쇄할 레벨을 동적으로 지정할 때 이 함수를 사용한다.

```
var x = 3;
printAsBitmapNum(x, "bmax");
```

위와 같은 코드 대신 printAsBitmap() 함수를 사용하면서 문자열을 합치는 방법을 사용해도 된다.

```
printAsBitmap("_level" + x, "bmax");
```

주의 사항

print() 함수의 주의 사항 참조

참조

print(), printAsBitmap(), printNum()

printNum() 전역 함수

특정 문서 레벨의 프레임을 벡터 형식으로 인쇄하는 메소드

버전 플래시 5

문법 printNum(level, boundingBox)

인자

level 인쇄할 문서 레벨을 나타내는 음이 아닌 정수 또는 음이 아닌 정수 값을 가지는 표현식

boundingBox

print()에서 설명한 것과 같이 level에 있는 프레임을 인쇄할 때 프레임의 자르는 방법을 나타내는 문자열

설명

printNum() 함수는 print() 함수와 거의 똑같지만 인쇄할 대상 레벨을 문자열이 아닌 숫자로 전달한다는 점에서 차이가 난다. 즉 printNum() 함수에서는 무비클립이 아니라 문서 레벨만 인쇄할 수 있다. 보통 다음과 같이 동적으로 문서의 레벨을 지정할 때 사용한다.

```
var x = 3;
printNum(x, "bmax");
```

위와 같은 코드 대신 print() 함수를 사용하면서 문자열을 합치는 방법도 사용해도 된다.

```
print("_level" + x, "bmax");
```

주의 사항

print() 함수의 주의 사항 참조

참조

print(), printAsBitmap(), printAsBitmapNum()

_quality 전역 속성

플레이어에서의 렌더링 화질

버전 플래시 5

문법 _quality

액세스 읽기/쓰기

설명

`_quality` 속성에는 다음과 같은 플래시 플레이어에서의 렌더링 화질을 결정하는 문자열이 저장된다.

“LOW” 저화질. 비트맵과 벡터 모두 안티앨리어싱(경계를 부드럽게 하는 것) 처리하지 않음

“MEDIUM” 보통 화질. 벡터 그래픽만 보통 수준으로 안티앨리어싱 처리함

“HIGH” 고화질. 벡터는 높은 수준으로 안티앨리어싱 처리하지만 비트맵은 고정된 그림만 안티앨리어싱 처리함

“BEST” 최고화질. 비트맵과 벡터 모두 높은 수준으로 안티앨리어싱 처리함

렌더링 화질을 낮출수록 한 프레임을 화면에 표시하는 데 필요한 계산 횟수가 줄어들기 때문에 속도가 빨라진다. 대부분 `_quality`는 “HIGH”로 설정한다.

예제

다음과 같이 하면 무비의 렌더링 화질을 최고화질로 설정할 수 있다(물론 이렇게 하면 재생 속도가 느려질 수 있다).

```
_quality = "BEST";
```

참조

```
_highQuality, toggleHighQuality()
```

random() 전역 함수

난수를 생성하는 함수

버전 플래시 4(플래시 5에서는 `Math.random()`을 사용함)

문법 `random(number)`

인자

number 양의 정수

리턴 값

0 이상, number 미만의 무작위로 발생시킨 정수

설명

이제는 더 이상 쓰이지 않는 random() 함수는 플래시 4에서 난수를 발생시키기 위한 용도로 사용된다. 이 함수는 더 이상 쓰이지 않으며 하위 호환성을 유지하기 위해 포함될 뿐이다. 플래시 5 이후 버전에서는 Math.random() 메소드를 사용하는 것이 좋다. random()에서는 0에서 number -1까지의 정수만을 난수로 발생시키지만 Math.random()에서는 0.0에서 0.999까지의 소수를 발생시킨다는 점에 주의하자.

참조

Math.random()

removeMovieClip() 전역 함수 주어진 무비 클립을 플래시 플레이어에서 제거하는 함수

버전 플래시 4(플래시 5에서는 attachMovie()를 이용하여 만든 함수도 제거할 수 있도록 개선됨)

문법 removeMovieClip(*target*)

인자

target 플레이어에서 제거할 무비 클립 인스턴스의 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 무비 클립 레퍼런스를 사용하면 경로로 자동 변환된다)

설명

removeMovieClip() 함수는 지정된 무비 클립을 플레이어에서 제거한다(이 때 클립의 내용이나 셀에는 흔적을 남기지 않는다). 그 뒤에 그 클립 또는 그 클립에 속하는 변수나 속성을 참조하면 undefined 값이 리턴된다.

removeMovieClip() 함수는 duplicateMovieClip()이나 attachMovie()를 이용하여 만든 무비 클립 인스턴스만 제거할 수 있다. 저작 도구에서 만든 무비 클립에 이 함수를 사용하면 아무런 일도 일어나지 않는다.

참조

`attachMovie()`, `duplicateMovieClip()`, `MovieClip.removeMovieClip()`; 13장의 ‘메인 무비 및 클립 인스턴스 제거’

_root 전역 속성

현재 레벨에 있는 무비의 메인 타임라인에 대한 레퍼런스

버전 플래시 5(플래시 4 무비에서의 “/”와 똑같다)

문법 `_root`

액세스 읽기 전용

설명

`_root` 속성에는 현재 문서 레벨에 있는 메인 타임라인에 대한 레퍼런스가 저장된다. `_root`는 메인 무비에 있는 메소드를 호출하거나 메인 무비 속성을 액세스하기 위한 용도로 쓰인다. 예를 들면 다음과 같다.

```
_root.play();           // 메인 타임라인을 재생한다.  
_root._alpha = 40;      // 무비 전체를 약간 투명하게 만든다.
```

`_root` 속성은 다음과 같은 식으로 중첩된 클립을 참조하는 데도 쓰인다.

```
_root.myClip.play();  
_root.shapes.square._visible = false;
```

`_root` 속성을 이용하면 무비 클립을 절대 경로 형식으로 액세스할 수 있다. 즉 `_root`로 시작하는 레퍼런스는 무비의 어느 위치에서 사용해도 된다.

참조

`_leveln`; `_parent`; 13장의 ‘`_root`와 `_leveln`을 이용하여 메인 무비 참조하기’와 2장의 ‘다른 타임라인에 있는 변수 사용하기’

scroll 속성

텍스트 필드에서 현재 표시된 첫 번째 줄

버전 플래시 4 이후**문법** `textfield.scroll`**리턴 값**

텍스트 필드에서 현재 표시된 첫 번째 줄의 번호를 나타내는 양의 정수

설명

scroll 텍스트 필드 속성은 읽기/쓰기 검용 속성이다. 텍스트 필드의 scroll 속성 값은 필드의 화면에 표시된 영역에서 첫 번째 줄에 해당하는 행 번호를 나타낸다. scroll 값을 새로 설정하면 텍스트 필드를 스크롤하여 주어진 번호의 줄이 텍스트 필드의 맨 윗줄이 되도록 만든다. scroll 속성은 보통 maxscroll과 함께 사용하며, '18 장. 온스크린 텍스트 필드'의 '텍스트 스크롤 코드'에서 설명한 텍스트 스크롤 인터페이스를 만들기 위한 용도로 쓰인다.

주의 사항

플래시에서는 scroll을 함수로 구분해 놓았지만, 사실 텍스트 필드 변수의 속성에 해당한다. scroll을 호출할 때는 괄호를 사용하지 않는다는 점에 주의하자.

버그

플래시 플레이어 5.0r30에서는 텍스트 필드에서 임베드된 글꼴을 사용하면 scroll을 사용했을 때 텍스트의 일부분이 필드의 가시 영역 밖에 표시되는 경우가 있다. 또한 어떤 텍스트는 필드를 스크롤할 때 제대로 지워지지 않고 그냥 남는 경우도 있다. 이러한 문제를 해결하려면 텍스트 필드 레이어 위에 마스크를 사용해야 한다. 이러한 문제점은 플래시 플레이어 5.0r41에서 수정되었다.

예제

```
// x에 myField에서 화면에 표시되는 첫 번째 행의 번호를 대입한다.
var x = myField.scroll;
// myField의 텍스트를 한 줄 아래로 스크롤한다.
myField.scroll++;
```

참조

maxscroll; 18장의 'scroll 속성'

Selection 객체

텍스트 필드 선택 영역을 제어하기 위한 객체

버전 플래시 5

문법 `Selection.methodName()`

메소드***getBeginIndex()***

선택된 첫 번째 글자의 인덱스를 리턴하는 메소드

getCaretIndex()

삽입 위치의 인덱스를 리턴하는 메소드

getEndIndex()

선택된 마지막 글자의 인덱스를 리턴하는 메소드

getFocus()

현재 삽입 위치가 있는 텍스트 필드를 리턴하는 메소드

setFocus()

특정 텍스트 필드로 삽입 위치를 이동하는 메소드

setSelection()

현재 선택된 텍스트 필드의 글자들을 선택하는 메소드

설명

Selection 객체는 텍스트 필드와 사용자간의 상호작용을 제어하고 텍스트 필드의 일부분을 캡처하기 위한 용도로 쓰인다. 실제 사용할 때는 Selection 객체 메소드 앞에 Selection이라는 키워드를 붙여야 한다. 이 메소드는 언제나 현재 포커스가 맞춰진 텍스트 필드에 대해서만 작동하며, 한 번에 하나씩의 텍스트 필드에만 포커스가 맞춰질 수 있다. Selection 객체의 메소드를 이용하면 삽입 위치를 정할 수 있고 텍스트 필드의 내용을 선택하거나 그 내용을 읽어올 수 있다. 이러한 특징을 활용하면 그다지 눈에 띄지는 않지만 사용자 입력 시스템에서 반드시 필요한 기능을

추가할 수 있다. 예를 들어 사용자 로그인 화면에서 이름을 입력하는 텍스트 필드로 커서를 이동시켜 사용자가 이름을 입력하도록 할 수 있다. 또는 문제가 되는 텍스트를 자동으로 선택하여 폼에서 고쳐야 할 부분을 강조할 수도 있다. 또한 'Selection.setFocus() 메소드'에 나온 것처럼 탭 순서를 조절할 수도 있다.

텍스트 필드에 있는 글자의 위치는 '첫 번째 글자가 0번, 두 번째 글자는 1번'과 같은 식으로 0부터 시작하는 인덱스를 이용하여 표시한다. 글자의 인덱스를 정하는 방법은 4장의 '문자 인덱스' 부분에서 자세히 설명한다.

플래시 5에서는 프로그램을 통해 텍스트를 잘라내거나 복사하거나 붙여넣을 수 없다. 또한 플래시 플레이어에서는 Ctrl-C나 Ctrl-V와 같은 잘라내기, 복사, 붙여넣기 단축키를 사용할 수 없다. 따라서 마우스의 보조 버튼(윈도우에서는 오른쪽 클릭, 매킨토시에서는 Ctrl-클릭)을 이용하여 잘라내기(cut), 복사(copy), 붙여넣기(paste) 기능을 사용해야 한다.

주의 사항

폼에 있는 보내기 버튼을 클릭하면 이전에 포커스가 설정된 텍스트 필드에서 자동으로 포커스가 사라진다. 포커스가 없어지기 전에 선택 영역을 캡처하고 싶다면 다음과 같이 버튼의 rollover 이벤트를 이용하면 된다.

```
on (rollover) {
    focusedField = Selection.setFocus();
}
```

참조

_focusrect, Selection.setFocus(); '18장. 온스크린 텍스트 필드'

Selection.getBeginIndex() 메소드

텍스트 필드에서 선택된 첫 번째 글자의 인덱스를 리턴하는 메소드

버전 플래시 5

문법 Selection.getBeginIndex()

리턴 값

현재 선택 영역(텍스트에서 강조된 부분)의 첫 번째 글자의 인덱스. 키보드 포커스가 설정된 텍스트 필드가 없으면 -1을 리턴한다. 텍스트 필드에 포커스가 설정되어 있긴 하지만 선택된 글자가 없으면 `Selection.getCaretIndex()`의 값을 리턴한다.

설명

`getBeginIndex()` 메소드는 선택 영역의 시작 부분을 구하기 위한 메소드이다. 현재 선택된 글자의 범위를 구하고 싶다면 `getBeginIndex()`와 `getEndIndex()`를 함께 사용하면 된다.

예제

아래 예제에서는 현재 선택된 글자들을 나타내는 문자열을 만들고 그 문자열을 Output 창에 표시한다.

```
var firstChar = Selection.getBeginIndex();
var lastChar  = Selection.getEndIndex();
var currentSelection = eval(Selection.getFocus()).substring(firstChar,
    lastChar);

trace(currentSelection);
```

다음과 같은 코드를 이용하면 선택 영역을 한 글자 왼쪽으로 확장할 수 있다.

```
Selection.setSelection(Selection.getBeginIndex() - 1,
    Selection.getEndIndex());
```

참조

`Selection.getCaretIndex()`, `Selection.getEndIndex()`

Selection.getCaretIndex() 메소드

삽입 위치의 인덱스를 리턴하는 메소드

버전 플래시 5

문법 `Selection.getCaretIndex()`

리턴 값

현재 텍스트 필드의 삽입 위치 인덱스. 포커스가 설정된 텍스트 필드가 없으면 -1을 리턴한다. 포커스가 설정된 텍스트 필드에 아무런 내용도 없으면 0을 리턴한다.

설명

getCaretIndex()는 텍스트 필드의 삽입 위치(커서가 있는 위치)를 리턴하는 메소드이다. 어떤 텍스트 필드에 포커스가 설정되어 있으면 커서가 I자 모양의 막대 형태가 된다. 삽입 위치를 설정할 때는 setSelection() 메소드를 이용하면 된다.

예제

포커스가 설정된 텍스트 필드가 없으면 getCaretIndex()에서 -1을 리턴하므로 getCaretIndex()의 값이 -1인지 확인하면 포커스가 설정된 텍스트 필드의 유무를 알아낼 수 있다.

```
if (Selection.getCaretIndex() == -1) {
    trace("No field has focus");
}
```

참조

Selection.setSelection()

Selection.getEndIndex() 메소드

선택된 마지막 글자의 인덱스를 리턴하는 메소드

버전 플래시 5

문법 Selection.getEndIndex()

리턴 값

현재 선택 영역(텍스트에서 강조된 블록)의 마지막 글자 다음 글자의 인덱스. 포커스가 설정된 텍스트 필드가 없으면 -1을 리턴한다. 포커스가 설정된 텍스트 필드는 있지만 아무 글자도 선택하지 않았을 때는 Selection.getCaretIndex()의 값을 리턴한다.

설명

`getEndIndex()`는 선택 영역의 마지막 글자를 알아내기 위한 메소드이다. 현재 선택된 글자의 범위를 알아낼 때는 `getEndIndex()`와 `getBeginIndex()`를 함께 사용하면 된다.

예제

아래 코드를 사용하면 선택 영역을 오른쪽으로 한 글자 확장시킬 수 있다.

```
Selection.setSelection(Selection.getBeginIndex(), Selection.getEndIndex() + 1);
```

참조

`Selection.getBeginIndex()`, `Selection.getCaretIndex()`

Selection.getFocus() 메소드

커서가 있는 텍스트 필드를 구하는 메소드

버전 플래시 5

문법 `Selection.getFocus()`

리턴 값

“_level1.myTextField”와 같이 키보드 포커스가 설정된(즉 커서가 있는) 텍스트 필드 변수에 대한 절대 경로를 나타내는 문자열. 키보드 포커스가 설정된 텍스트 필드가 없으면 null을 리턴한다.

설명

`getFocus()`는 현재 포커스가 설정된 텍스트 필드의 경로를 나타내는 문자열을 리턴하는 메소드이다. 이 문자열을 변수 레퍼런스로 변환할 때는 `eval()`을 이용하면 된다. 예를 들어 아래 코드에서는 포커스가 설정된 텍스트 필드에 있는 글자의 개수를 알아낼 수 있다. `getFocus()`를 이용하여 필드의 이름을 알아내고 `eval()`을 이용하여 그 이름을 변수로 변환한다.

```
var numChars = eval(Selection.getFocus()).length;
```

예제

포커스가 설정된 텍스트 필드가 없으면 `getFocus()`에서 `null`을 리턴하는 성질을 이용하여 다음과 같이 `getFocus()`와 `null`을 비교하면 포커스가 설정된 필드가 있는지 알아낼 수 있다.

```
if (Selection.getFocus() == null) {
    trace("No field has focus");
}
```

참조

`Selection.setFocus()`, `eval()`, `Selection.getCaretIndex()`

Selection.setFocus() 메소드

특정 텍스트 필드에 키보드 포커스를 설정한다.

버전 플래시 5

문법 `Selection.setFocus(variableName)`

인자

variableName

포커스를 설정할 텍스트 필드 변수를 나타내는 문자열 경로(예: “_root.myTextField” 또는 “userName”)

리턴 값

포커스를 설정하는 데 성공했는지(true) 실패했는지(false) 나타내는 부울 값. `variableName`으로 지정한 변수가 없거나 그 변수가 텍스트 필드 변수가 아닌 경우에만 포커스를 설정할 수 없다.

설명

`setFocus()`는 텍스트 필드에 대해 키보드 포커스를 설정하는 메소드이다. 이 메소드에서는 지정된 텍스트 필드로 커서를 이동하며, 보통 사용자가 그 필드에 집중하도록 하거나 데이터를 입력하도록 할 때 이러한 방법을 사용한다. `setFocus()` 메소드를 이용하면 사용자가 실수로 잘못된 내용을 입력한 부분을 알려줄 수 있다. 온라인 폼에서 사용자가 이메일 주소를 잘못 입력한 경우를 생각해 보자. 사용자에게

이러한 오류를 알리고 싶다면 이메일 주소를 입력하는 텍스트 필드에 포커스를 설정하여 잘못된 주소를 고치도록 할 수 있다. 또한 뒤에 나오는 예제에 나온 것과 같이 탭 순서를 원하는 대로 설정할 수도 있다.

주의 사항

텍스트 필드에 포커스를 설정하면 그 필드에 있는 모든 내용이 자동으로 선택된다는 점에 주의하자. 텍스트 필드의 내용을 선택하지 않으면서 포커스를 설정하고 싶다면 다음과 같은 코드를 이용해야 한다.

```
// 우선 myField에 포커스를 설정한다.
Selection.setFocus("myField");

// 삽입 위치를 myField의 시작 부분으로 옮긴다.
Selection.setSelection(0, 0);

// 또는 다음과 같이 myField의 끝 부분으로 삽입 위치를 옮길 수도 있다.
Selection.setSelection(myField.length, myField.length);
```



무비를 브라우저에서 재생할 때는 플래시 무비 자체에 포커스가 설정되어 있어야만(즉 사용자가 무비가 재생되는 동안에 무비를 클릭해야만) 포커스가 설정된 텍스트 필드에 텍스트를 입력할 수 있다. 텍스트 필드에 어떤 내용을 입력해달라는 요청을 하기 전에 무비 자체에 포커스가 설정되어 있는지 확인하도록 하자. 무비 시작 부분에 “Click Here to Start”와 같은 버튼을 추가하는 것도 좋은 방법이다.

예제

아래 예제는 사용자 입력 폼에 있는 필드의 탭 순서를 설정하는 방법을 보여준다. 이 코드가 들어있는 샘플 .fla 파일은 온라인 코드 창고에서 다운로드할 수 있다(탭 키를 다루는 방법에 대한 자세한 정보는 ‘10장. 이벤트 및 이벤트 핸들러’의 ‘특수 키 처리’ 부분에 나와있다).

```
// 사용자 정의 탭 순서
// 폼 필드가 들어있는 무비 클립에 들어가는 코드
onClipEvent (load) {
    // 이 클립의 경로를 문자열로 저장한다. 이 값은 나중에
    // Selection.setFocus()를 호출할 때 사용한다.
    path = targetPath(this);
    // 원하는 탭 순서대로 텍스트 필드의 이름을 저장해 둔 배열을
```

```

// 만든다. 첫 번째 텍스트 필드가 기본으로 선택된다.
tabOrder = ["field1", "field3", "field2", "field4"];
}

onClipEvent (keyUp) {
    // 탭 키를 누르면...
    if (Key.getCode() == Key.TAB) {
        // 어떤 필드에도 포커스가 설정되어 있지 않을 때
        if (Selection.getFocus() == null) {
            // 필드 배열에 있는 첫 번째 필드에 포커스를 설정한다.
            Selection.setFocus(path + "." + tabOrder[0]);
        } else {
            // 그렇지 않으면 필드 배열에서 현재 포커스가 설정된 필드를 찾는다.
            i = 0;
            focused = Selection.getFocus();
            while (i < tabOrder.length) {
                // Selection.getFocus()에서 리턴한 전체 경로로부터
                // 필드 변수의 이름을 뽑아낸다.
                fieldName = focused.substring(focused.lastIndexOf(".") + 1);
                // tabOrder의 각 원소를 돌면서 포커스가 설정된 필드를 찾는다.
                if (tabOrder[i] == fieldName) {
                    // 필드 이름을 찾으면 순환문을 끝낸다.
                    currentFocus = i;
                    break;
                }
                i++;
            }
            // tabOrder 배열에 있는 필드 중 포커스가 설정된 변수를 찾았으므로
            // 시프트 키를 눌렀는지 확인해 보고 그에 맞게 포커스를 다음 필드
            // 또는 그 앞의 필드에 맞춘다.
            if (Key.isDown(Key.SHIFT)) {
                // 시프트 키가 눌러있는 상태이므로 다음 필드에 포커스를 설정한다.
                // 현재 필드가 첫 번째 필드라면 마지막 필드로 돌아가도록 한다.
                nextFocus = currentFocus-1 == -1 ? tabOrder.length-1 :
                    currentFocus-1;
            } else {
                // 시프트 키를 누르지 않은 경우. 마지막 필드에서는 첫 번째 필드로,
                // 그렇지 않으면 다음 필드로 포커스를 넘긴다.
                nextFocus = currentFocus+1 == tabOrder.length ? 0 :
                    currentFocus+1;
            }
            // 새로운 포커스를 설정한다.

```

```
        Selection.setFocus(path + "." + tabOrder[nextFocus]);
    }
}
}

// 메인 타임라인에 있는 버튼에 들어갈 코드
on (keyPress "<Tab>") {
    // 이 코드는 인터넷 익스플로러에서 탭 키를 잡아내기 위한 용도로 쓰인다.
    var tabCatcher = 0;
}
```

참조

Selection.setFocus(), Selection.setSelection()

Selection.setSelection() 메소드 포커스가 설정된 텍스트 필드의 글자들을 선택하거나
그 위치에 삽입 위치를 설정하는 메소드

버전 플래시 5

문법 Selection.setSelection(*beginIndex*, *endIndex*)

인자

beginIndex 새로운 선택 영역의 첫 번째 글자의 인덱스를 나타내는 0 이상의 정수

endIndex 새로운 선택 영역의 마지막 글자 다음 글자의 인덱스를 나타내는 0 이상의 정수

설명

setSelection() 메소드에서는 포커스가 설정된 텍스트 필드에서 *beginIndex*부터 *endindex*-1까지의 글자들을 선택(강조)한다. 포커스가 설정된 필드가 없다면 아무런 효과도 나타나지 않는다. 보통 사용자가 입력한 내용 중에서 잘못된 부분을 강조할 때 이 메소드를 사용한다.

주의 사항

Selection 객체에는 setCaretIndex와 같이 포커스가 설정된 필드에서 삽입 위치를 설정하는 메소드는 없지만 setSelection() 메소드를 이용하면 텍스트 필드의 특

정 위치에 삽입 위치를 설정할 수 있다. 다음과 같이 `beginIndex`와 `endIndex`에 같은 값을 전달하기만 하면 된다.

```
// 세 번째 글자 뒤에 삽입 위치를 설정한다.
Selection.setSelection(3, 3);
```

예제

```
// 현재 포커스가 설정된 텍스트 필드의
// 두 번째와 세 번째 글자를 선택한다.
Selection.setSelection(1, 3);
```

참조

`Selection.getBeginIndex()`, `Selection.getCaretIndex()`, `Selection.getEndIndex()`

setProperty() 전역 함수

무비 클립 속성에 새로운 값을 대입한다.

버전 플래시 4 이후

문법 `setProperty(movieClip, property, value)`

인자

movieClip 무비 클립의 경로를 나타내는 문자열 표현식. 플래시 5에서는 문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 자동 변환되므로 이 자리에 무비 클립 레퍼런스를 사용해도 된다.

property value를 대입할 내장 속성의 이름. 문자열이 아니라 인식자를 전달해야 한다(예: “_alpha”가 아니라 _alpha를 사용해야 한다).

value movieClip의 property 속성에 대입할 새로운 데이터 값

설명

`setProperty()` 함수는 movieClip의 내장 속성(내장 속성 목록은 MovieClip 클래스를 설명하는 부분에 나와 있다)에 value 값을 대입하는 함수이다. 사용자 정의 속성 값을 설정할 때는 이 함수를 사용할 수 없다. 플래시 4에서는 무비 클립 속성에

새로운 값을 대입할 때 `setProperty()` 함수만을 사용해야 했다. 하지만 플래시 5 이후 버전에서는 내장 및 사용자 정의 무비 클립 속성에 새로운 값을 대입할 때 . 연산자와 [] 연산자를 이용하는 것이 좋다.

예제

```
// 플래시 4에서 사용하는 방법. 메인 무비를 45도 회전시킨다.  
setProperty("_root", _rotation, 45);
```

```
// 플래시 5에서 사용하는 방법. 마찬가지로 메인 무비를 45도 회전시킨다.  
_root._rotation = 45;
```

참조

`getProperty()`; 13장의 '무비 클립의 객체성', 부록 C

Sound 클래스

무비의 사운드를 제어하기 위한 클래스

버전 플래시 5

생성자 `new Sound()`
 `new Sound(target)`

인자

target 사운드를 제어할 무비 클립 또는 문서 레벨의 경로를 나타내는 문자열. 무비 클립이나 문서 레벨에 대한 레퍼런스를 이 인자로 사용해도 된다(문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 자동 변환되기 때문이다).

메소드

attachSound() 라이브러리의 사운드를 Sound 객체와 연결하는 메소드

getPan() 현재 팬 설정을 구하는 메소드

getTransform() 왼쪽과 오른쪽 스피커의 사운드 채널 분포(밸런스)를 구하는 메소드

getVolume()

음량을 알아내는 메소드

setPan()

사운드의 왼쪽과 오른쪽 채널에 대한 팬을 설정하는 메소드

setTransform()

왼쪽과 오른쪽 스피커의 사운드 채널 분포(밸런스)를 설정하는 메소드

setVolume() 사운드 음량을 설정하는 메소드***start()*** 사운드를 재생하는 메소드***stop()*** 전체 사운드 또는 특정 사운드만 중지시키는 메소드**설명**

Sound 클래스의 객체는 무비의 사운드를 제어하거나 프로그램을 통해 추가한 사운드를 제어하기 위한 용도로 쓰인다. Sound 객체에는 몇 가지 응용 방법이 있다. 이 객체에서는 다음과 같은 사항을 제어할 수 있다.

- 플래시 플레이어의 모든 사운드
- 특정 무비 클립 인스턴스나 메인 무비의 모든 사운드(그 밑에 중첩된 클립의 사운드 포함)
- 프로그램을 통해 추가한 개별 사운드

플래시 플레이어의 모든 사운드(문서 레벨의 .swf 파일에 있는 사운드 포함)를 제어할 수 있는 Sound 객체를 만들려면 인자 없이 생성자를 호출하면 된다.

```
myGlobalSound = new Sound();
```

특정 클립이나 메인 무비에 있는 모든 사운드를 제어하는 Sound 객체를 만들 때는 생성자를 호출할 때 제어할 클립이나 무비를 나타내는 target 매개변수를 전달하면 된다. 이렇게 하면 target에 포함된 모든 클립의 사운드를 제어할 수 있다.

```
spaceshipSound = new Sound("spaceship"); // spaceship 클립의
                                           // 사운드 제어
mainSound      = new Sound("_root");     // 메인 타임라인의
                                           // 사운드 제어
```

독립적으로 재생하거나 정지시키거나 순환시킬 수 있는 사운드를 만들고 싶다면 Sound 객체를 만들고 나서 `attachSound()` 메소드를 이용하여 사운드를 추가하면 된다.

참조

`stopAllSounds(); _soundbuftime`

Sound.attachSound() 메소드 라이브러리에 있는 사운드를 Sound 객체에 연결시키는 메소드

버전 플래시 5

문법 `soundObject.attachSound(linkageIdentifier)`

인자

linkageIdentifier

추가할 사운드의 이름(라이브러리의 Options → Linkage에서 지정함)

설명

`attachSound()`는 무비 재생 중에 새로운 사운드를 추가하고 그 사운드를 `soundObject`로 제어할 수 있도록 해주는 메소드이다. 일단 사운드를 추가하고 나면 `soundObject`의 `start()`와 `stop()` 메소드를 이용하여 그 사운드를 독립적으로 재생시키거나 중지시킬 수 있다.

사운드를 `soundObject`에 추가하려면 무비의 라이브러리에서 그 사운드를 내보내야 한다. 사운드를 내보낼 때는 다음과 같은 단계를 거치면 된다.

1. 라이브러리에서 내보낼 사운드를 선택한다.
2. Options → Linkage를 선택한다. Symbol Linkage Properties 대화상자가 나타난다.
3. Export This Symbol을 선택한다.
4. Identifier 부분에 그 사운드의 이름을 입력한다.

플래시 파일에 포함시킨 사운드는 모두 그 사운드를 포함하는 무비의 첫 번째 프레임에서 로딩되므로(액션스크립트에서 그 사운드를 처음 사용할 때 로딩되지 않는다) 사운드 용량이 크면 로딩하는 데 시간이 오래 걸린다는 점에 주의하자. 각 사운드를 외부 .swf 파일로 저장하고 필요할 때 loadMovie()를 이용하여 그 사운드를 불러오면 사운드를 로딩하는 데 필요한 시간을 적절히 조절할 수 있다.

주의 사항

Sound 객체에는 한 번에 하나의 사운드만 집어넣을 수 있다. Sound 객체에 새로운 사운드를 추가하면 이전에 그 객체에 들어있던 사운드는 제거된다. loadMovie()를 이용하여 어떤 클립이나 레벨로 불러온 무비에 대해서는 추가된 사운드가 Sound 객체의 영역에 해당되는 문서의 라이브러리에 있지 않은 이상 attachSound()를 사용할 수 없다. target 매개변수를 사용하지 않고 만든 전역 Sound 객체의 영역은 _level0이 된다.

예제

아래 예제에서는 phaser라는 사운드를 phaserSound라는 Sound 객체에 추가한다. 그리고 phaser 사운드를 재생하고 나서 멈춘다.

```
phaserSound = new Sound();
phaserSound.attachSound("phaser");

// phaser 사운드를 재생한다.
phaserSound.start();

// phaser 사운드 재생을 멈춘다.
phaserSound.stop("phaser");
```

참조

Sound.start(), Sound.stop()

Sound.getPan() 메소드

가장 최근의 팬 값을 구하는 메소드

버전 플래시 5**문법** `soundObject.getPan()`**리턴 값**

`setPan()`으로 설정한 가장 최근의 팬 값을 나타내는 숫자. 보통 -100(왼쪽 채널만 나오고 오른쪽 채널은 전혀 나오지 않음)에서 100(오른쪽 채널만 나오고 왼쪽 채널은 전혀 나오지 않음) 사이의 값을 가진다. 기본값은 0(왼쪽과 오른쪽 채널의 음량이 같음)이다.

설명

사운드의 팬(pan)을 조절하면 음원의 위치가 움직이는 것과 같은 효과를 낼 수 있다. `getPan()`를 이용하면 `soundObject`에서 제어하는 사운드의 왼쪽 채널과 오른쪽 채널의 분포를 알아낼 수 있다. 일반적으로 `getPan()`은 `setPan()`과 함께 사운드의 팬을 조절하는 데 사용된다.

예제

사운드의 팬을 20 줄인다.

```
mySound = new Sound();
mySound.setPan(mySound.getPan() - 20);
```

참조

`Sound.getTransform()`, `Sound.setPan()`

Sound.getTransform() 메소드

왼쪽과 오른쪽 스피커의 사운드 채널 분포를 알아내는 메소드

버전 플래시 5**문법** `soundObject.getTransform()`**리턴 값**

`soundObject`로 제어하는 사운드의 채널 백분율 값을 속성으로 하는 익명 객체

설명

`getTransform()` 메소드에서는 `soundObject`에서 제어하는 사운드의 채널이 왼쪽과 오른쪽 스피커에 어떤 식으로 분포되어 있는지를 알려주는 값들을 속성으로 가지는 객체를 리턴한다. 리턴된 객체의 속성은 `ll`, `lr`, `rl`, `rr`이며, 각 속성에 대한 설명은 `Sound.setTransform()` 메소드를 설명하는 부분에 나와있다.

참조

`Sound.getPan()`, `Sound.setTransform()`

Sound.getVolume() 메소드

현재 음량 설정을 구하는 메소드

버전 플래시 5

문법 `soundObject.getVolume()`

리턴 값

`setVolume()`에서 설정한 현재 음량을 나타내는 숫자. 보통 0에서 100(기본 음량)까지의 값을 갖지만 더 높은 값을 사용할 수도 있다.

설명

`getVolume()`는 `soundObject` 객체에서 제어하는 사운드의 음량을 리턴하는 메소드이다. 일반적으로 `getVolume()`은 `setVolume()`과 함께 사운드의 음량을 조절하기 위한 용도로 쓰인다.

예제

아래 코드에서는 사운드 음량을 20 줄인다.

```
mySound = new Sound();
mySound.setVolume(mySound.getVolume() - 20);
```

참조

`Sound.setVolume()`

Sound.setPan() 메소드

사운드의 왼쪽과 오른쪽 채널의 밸런스를 조절하는 메소드

버전 플래시 5**문법** `soundObject.setPan(pan)`**인자**

pan soundObject 객체에서 제어하는 사운드의 왼쪽과 오른쪽 스피커의 음량 분포를 나타내는 -100(왼쪽) 이상 100(오른쪽) 이하의 숫자. pan 인자의 값이 100보다 크면 $200 - \text{pan}$ 이, -100보다 작으면 $-200 - \text{pan}$ 이 pan 대신 쓰인다.

설명

setPan() 메소드는 soundObject에서 제어하는 사운드의 왼쪽과 오른쪽 채널 사이의 밸런스를 조절하는 기능을 제공한다. 시간이 지남에 따라 팬(pan) 값을 조절하면 한쪽 스피커에서 다른쪽 스피커로 소리가 움직이는 것과 같은 효과(패닝(panning)이라고 부름)을 나타낼 수 있다.

soundObject에서 제어하는 사운드를 왼쪽 스피커에서만 나오도록 하려면 pan 값을 -100으로, 오른쪽 스피커에서만 나오도록 하려면 100으로 설정하면 된다. 양쪽 채널의 음량을 똑같이 맞출 때는 pan 값을 0으로 설정하면 된다.

setPan()은 soundObject에서 제어하는 모든 사운드에 영향을 미친다는 점에 주의하자. soundObject가 전역 사운드이면 setPan() 메소드를 사용했을 때 무비 전체의 사운드가 영향을 받는다. soundObject가 특정 클립이나 메인 타임라인에 연결되어 있다면 해당 클립이나 타임라인 전체의 사운드가 영향을 받는다.

setPan()으로 팬을 설정하고 나서 팬을 다시 설정하고 싶다면, setPan()을 다시 호출해야 한다. setPan() 메소드로 일단 좌우 밸런스를 설정하고 나면 soundObject가 제거된 후에도 그 효과가 지속된다.

예제

다음과 같은 클립 이벤트 핸들러를 이용하면 무비 클립의 사운드가 왼쪽과 오른쪽 스피커 사이에서 끊임없이 패닝된다.

```

onClipEvent (load) {
    panEffect = new Sound(this);
    panDirection = "right";
    panIncrement = 50;
}

onClipEvent (enterFrame) {
    if (panDirection == "right") {
        newPan = panEffect.getPan() + panIncrement;
        if (newPan > 100) {
            panDirection = "left";
            panEffect.setPan(panEffect.getPan() - panIncrement);
        } else {
            panEffect.setPan(newPan);
        }
    } else {
        newPan = panEffect.getPan() - panIncrement;
        if (newPan < -100) {
            panDirection = "right";
            panEffect.setPan(panEffect.getPan() + panIncrement);
        } else {
            panEffect.setPan(newPan);
        }
    }
}
}

```

다음과 같은 클립 이벤트 핸들러를 이용하면 마우스 움직임에 따라 사운드가 달라진다. 이 코드는 스테이지 너비와 높이가 각각 550과 400인 경우를 기준으로 작성된 것이며, 마우스의 수평방향 위치에 따라 사운드를 왼쪽이나 오른쪽으로 이동시키고 마우스의 수직방향 위치에 따라 음량을 높이거나 줄인다.

```

onClipEvent (load) {
    // 새로운 Sound 객체를 만들고 bgMusic 사운드를 그 객체에 추가한다.
    mySound = new Sound(this);
    mySound.attachSound("bgMusic");
    mySound.start(0, 999);    // 사운드를 재생시킨다(계속 반복시킴).
}

onClipEvent (enterFrame) {
    // 마우스의 수평 위치를 구해서 그에 따라 팬을 설정한다.
    mouseX = (_root._xmouse / 550) * 200;

```



```
mySound.setPan(mouseX - 100);  
// 마우스의 수직 위치를 구해서 그 값에 따라 음량을 조절한다.  
mouseY = (_root._ymouse / 400) * 300;  
mySound.setVolume(300 - mouseY);  
}
```

참조

Sound.getPan()

Sound.setTransform() 메소드

좌우 채널을 좌우 스피커에 분포시키는 메소드

버전 플래시 5

문법 `soundObject.setTransform(transformObject)`

인자

transformObject

일련의 속성을 이용하여 새로운 채널 설정을 나타내는 사용자 정의 객체

설명

setTransform() 메소드를 이용하면 사운드의 채널을 좌우 스피커를 통해 출력하는 방법을 정밀하게 조절할 수 있다. 이론적으로 보면 setTransform()과 setPan()은 서로 다를 것이 없지만 스테레오 사운드를 조절할 때 더 자세하게 채널 분포를 설정할 수 있다.

스테레오 사운드는 두 가지 서로 다른 사운드(왼쪽 채널과 오른쪽 채널)의 조합으로 이루어지며 이 두 사운드는 보통 각각 왼쪽과 오른쪽 스피커로 출력된다. 하지만 setTransform()을 이용하면 각 채널을 얼마큼씩 각 스피커로 보낼지 결정할 수 있다. 예를 들어 '왼쪽 채널의 절반만 왼쪽 스피커로 보내고 오른쪽 채널은 두 스피커에 모두 보낸다' 또는 '왼쪽과 오른쪽 채널을 모두 왼쪽 스피커에서만 재생한다'와 같은 식으로 채널 분포를 조절할 수 있다.

setTransform()을 이용하려면 우선 미리 정의된 일련의 속성을 포함하고 있는 객체를 만들어야 한다. 이 객체의 속성을 이용하여 왼쪽과 오른쪽 스피커에 왼쪽과 오른쪽 채널을 출력하는 방법을 지정할 수 있다.

[표 R-12] transformObject의 속성

속성 이름	속성 값	속성 설명
ll	0 ~ 100	왼쪽 스피커에서 재생할 왼쪽 채널의 비율
lr	0 ~ 100	왼쪽 스피커에서 재생할 오른쪽 채널의 비율
rl	0 ~ 100	오른쪽 스피커에서 재생할 왼쪽 채널의 비율
rr	0 ~ 100	오른쪽 스피커에서 재생할 오른쪽 채널의 비율

[표 R-12]에 나온 것과 같은 속성을 가진 객체를 만들고 나면 그 객체를 Sound 객체의 setTransform() 메소드에 전달하면 된다. transformObject의 속성에 의해 soundObject에서 제어하는 사운드의 채널 출력 백분율이 결정된다.

특정 Sound 객체의 현재 설정을 확인하고 싶다면 getTransform() 메소드를 호출하면 된다.

예제

```
// 새로운 Sound 객체를 만든다.
mySound = new Sound();

// setTransform()에 전달할 객체를 만든다.
transformer = new Object();

// transformer 객체의 속성을 설정한다.
transformer.ll = 0; // 왼쪽 스피커에서 재생할 왼쪽 채널의 비율(0%)
transformer.lr = 0; // 왼쪽 스피커에서 재생할 오른쪽 채널의 비율(0%)
transformer.rl = 0; // 오른쪽 스피커에서 재생할 왼쪽 채널의 비율(0%)
transformer.rr = 100; // 오른쪽 스피커에서 재생할 오른쪽 채널의 비율(100%)

// transformer 객체를 setTransform() 메소드에 전달하여
// 새로운 채널 분포를 적용한다.
mySound.setTransform(transformer);
```

참조

Sound.getTransform(), Sound.setPan()

Sound.setVolume() 메소드

Sound 객체에서 제어하는 사운드의 음량을 설정하는 메소드

버전 플래시 5

문법 *soundObject.setVolume(volume)*

인자

volume soundObject에서 제어하는 사운드의 음량을 나타내는 숫자. 0으로 설정하면 아무 소리도 나지 않는다. volume의 절대값(부호는 상관없음)이 클수록 soundObject에서 제어하는 사운드 크기가 커진다. 예를 들어 volume 값이 -50인 경우와 50인 경우의 소리 크기는 똑같다. volume의 기본값은 100이다.

설명

setVolume()은 soundObject에서 제어하는 사운드의 크기를 조절하는 메소드이다. 소리가 전혀 나지 않도록 하려면 volume 값을 0으로 설정하면 된다. 소리를 더 크게 하려면 volume 값을 키우면 된다. 100에서 200 정도의 값을 사용하면 꽤 큰 소리가 나긴 하지만 최대값이 정해져 있지는 않다.

setVolume() 메소드를 이용하면 soundObject에서 제어하는 모든 사운드가 영향을 받는다는 점에 주의하자. soundObject가 전역 사운드이면 setVolume()에 의해 무비의 사운드 전체가 영향을 받는다. soundObject가 특정 클립이나 메인 타임라인에 연결되어 있으면 그 클립이나 타임라인의 사운드 전체가 영향을 받는다.

setVolume()의 효과는 다음 번 setVolume()을 호출하기 전까지 지속된다. 한번 setVolume() 메소드로 음량을 설정하고 나면 그 뒤로는 soundObject에서 제어하는 모든 사운드가 영향을 받게 되며, soundObject가 제거되고 난 뒤에도 그 영향이 지속된다.

예제

다음 코드에서는 무비 클립의 음량을 조절한다.

```
var mySound = new Sound();
mySound.setVolume (65);
```

무비의 음량 레벨을 조절하기 위한 버튼은 다음과 같은 식으로 만들 수 있다.

```
// 메인 무비 타임라인에 들어갈 코드
var globalSound = new Sound();
var maxVolume = 200;
var minVolume = 0;
var volumeIncrement = 20;

// 메인 타임라인의 음량을 키우는 버튼에 들어갈 코드
on (release) {
    globalSound.setVolume(Math.min(globalSound.getVolume()
                                    + volumeIncrement, maxVolume));
}

// 메인 타임라인의 음량을 줄이는 버튼에 들어갈 코드
on (release) {
    globalSound.setVolume(Math.max(globalSound.getVolume()
                                    - volumeIncrement, minVolume));
}
```

참조

Sound.getVolume()

Sound.start() 메소드

사운드 재생을 시작하는 메소드

버전 플래시 5

문법 *soundObject.start(secondOffset, loops)*

인자

secondOffset

soundObject의 사운드를 재생할 시점을 초 단위로 나타내는 부동소수점 소수(시작 지점(entry point)이라고도 부름). 예를 들어 secondOffset을 1로 설정하면 라이브러리에 들어 있는 사운드의 1초 지점부터 사운드를 재생한다. 기본값은 0이다. 사운드 재생을 중지하

는 시점을 나타내는 값은 정해져 있지 않다. 사운드는 수동으로 중지시키지 않으면 사운드가 끝날 때까지 재생된다.

loops soundObject의 사운드를 반복할 회수를 나타내는 양의 정수. 사운드를 한 번만 재생하려면 1(기본값)을, 두 번 연속으로 재생하려면 2를 대입하면 된다. 따라서 secondOffset부터 시작되는 부분을 loops로 지정한 횟수만큼 반복해서 재생한다.

설명

start()는 attachSound()를 이용하여 soundObject에 추가한 프로그램을 통해 정의한 사운드를 재생하기 위한 메소드이다. start() 메소드에서는 클립이나 무비의 모든 사운드를 재생하는 것은 아니고 가장 최근에 attachSound()를 이용하여 soundObject에 추가한 사운드만 재생한다.

soundObject의 사운드의 일부분만 재생하는 경우에는 secondOffset 인자를 이용하면 된다. 또한 여러 번 반복해서 재생할 때는 loops 인자를 이용하면 된다.

예제

```
// 새로운 Sound 객체를 만든다.
boink = new Sound();

// boink라는 이름으로 저장된 사운드를 Sound 객체에 추가한다.
boink.attachSound("boink");

// boink 전체를 재생한다. soundOffset은 기본값인 0이 된다.
boink.start();

// 0.5초 지점부터 boink를 재생한다. loops는 기본값인 1이 된다.
boink.start(.5);

// boink를 처음부터 끝까지 세 번 반복한다.
boink.start(0, 3);
```

참조

Sound.stop()

Sound.stop() 메소드

모든 사운드 또는 특정 사운드를 정지시키는 메소드

버전 플래시 5

문법 `soundObject.stop()`
`soundObject.stop(linkageIdentifier)`

인자***linkageIdentifier***

soundObject와 같은 target을 가지는 사운드의 이름. 연결 인식자(linkage identifier)는 라이브러리의 Options → Linkage에서 지정한다.

설명

linkageIdentifier를 지정하지 않고 stop()을 호출하면 soundObject에서 제어하는 모든 사운드를 중지시킨다. soundObject가 전역 사운드이면 무비에 있는 모든 사운드가 중지되고 target 매개변수를 지정하여 soundObject를 만든 경우에는 target의 모든 사운드가 중지된다.

linkageIdentifier 인자를 사용하면 그 이름을 가지는 특정 사운드만 정지시킨다. 이 때 linkageIdentifier는 attachSound()를 이용하여 Sound 객체에 추가한 사운드를 가리키는 인식자이다. 정지시킬 사운드가 반드시 soundObject 자체여야 하는 것은 아니다. soundObject와 같은 target을 공유하는 임의의 Sound 객체는 모두 정지된다. 만약 soundObject를 만들 때 target 인자를 사용하지 않았다면(즉 전역 Sound 객체라면) 다른 전역 Sound 객체의 사운드를 정지시킬 수도 있다.

예제

```
// 전역 Sound 객체를 만든다.
mySound = new Sound();

// doorbell이라는 사운드를 객체에 추가한다.
mySound.attachSound("doorbell");

// 무비의 모든 사운드를 정지시킨다.
mySound.stop();
```

```
// doorbell을 재생한다.  
mySound.start();  
  
// doorbell만 재생한다.  
mySound.stop("doorbell");  
  
// 다른 전역 Sound 객체를 만든다.  
myOtherSound = new Sound();  
  
// 그 객체에 doorknock 사운드를 추가한다.  
myOtherSound.attachSound("doorknock");  
  
// doorknock을 재생한다.  
myOtherSound.start();  
  
// myOtherSound가 아니라 mySound를 통해서 doorknock을 재생한다.  
// 두 개의 사운드 객체의 대상이 같기 때문에 이렇게 해도 된다.  
mySound.stop("doorknock");
```

참조

Sound.start()

_soundbuftime 전역 속성

미리 불러올 스트리밍 사운드 길이(초 단위)

버전	플래시 4 이후
문법	<code>_soundbuftime</code>
액세스	읽기/쓰기

설명

`_soundbuftime` 속성은 사운드를 재생하기 전에 미리 불러올 분량을 초 단위로 나타낸 정수이다. 기본값은 5초이다.

플래시에서는 만화 캐릭터의 입술과 대사가 잘 맞을 수 있도록 무비를 스트리밍 사운드와 동기화시킨다. `_soundbuftime`에서 지정한 시간만큼의 사운드를 미리 불러올 때까지 애니메이션이 중단되므로, 이 값을 너무 크게 잡으면 네트워크가 느린 경우에 무비를 시작하는 데 시간이 너무 오래 걸릴 수도 있다. 네트워크 스트리밍의

속도가 느리거나 가끔씩 접속이 끊길 때 `_soundbuftime` 설정을 너무 짧게 하면 소리가 끊길 수도 있다(사운드 데이터가 충분히 버퍼링되지 않는 경우). 이상적인 `_soundbuftime` 값은 그래픽의 복잡한 정도, 음질, 사용자의 인터넷 접속 속도와 같은 요소에 따라 달라질 수 있다. 기본값(5초)을 사용해도 괜찮지만 각 경우에 따른 최적의 조건을 찾으려면 직접 여러 번 시도해 보는 수밖에 없다. 재생 도중에 `_soundbuftime`을 변경하는 것도 가능하지만, 이 속성은 전역 속성이기 때문에 각 사운드별로 다른 값을 적용할 수는 없다.

예제

```
_soundbuftime = 10; // 10초 분량을 미리 버퍼링한다.
```

startDrag() 전역 함수

무비나 무비 클립이 마우스 포인터를 따라가도록 하는 메소드

버전 플래시 4 이후

문법

```
startDrag(target)
startDrag(target, lockCenter)
startDrag(target, lockCenter, left, top, right, bottom)
```

인자

target 마우스 포인터를 따라 움직일 무비 또는 무비 클립의 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 무비 클립 레퍼런스를 사용하면 경로로 자동 변환된다)

lockCenter target의 등록 지점을 마우스 포인터 위치(true)에 고정시킬지 원래 위치에 대해 상대적으로(false) 움직일지를 나타내는 부울 값

left target의 등록 지점이 움직일 수 있는 x의 최소값(제일 왼쪽)

top target의 등록 지점이 움직일 수 있는 y의 최대값(위쪽)

right target의 등록 지점이 움직일 수 있는 x의 최대값(제일 오른쪽)

bottom target의 등록 지점이 움직일 수 있는 y의 최소값(아래쪽)

설명

`startDrag()`는 `target`이 플레이어 안에서 마우스 포인터를 따라 움직이게 하는 함수이다(클립을 드래그한다고 부른다). 이렇게 클립을 드래그할 때는 `startDrag()` 함수에 전달하는 인자로 정의되는 경계 상자 안에서만 움직일 수 있다. 경계 상자의 좌표는 `target`이 들어있는 캔버스의 좌표를 기준으로 계산한다. 캔버스가 메인 무비 스테이지이면 스테이지의 왼쪽 위 구석이 (0, 0)점이 된다. 만약 캔버스가 무비 클립이라면 클립 캔버스의 등록 지점이 (0, 0)이 된다. 플래시의 좌표계는 직각 좌표계에서 Y축을 뒤집어 놓은 형태이다. 즉 화면 아래쪽으로 가면 y 값이 증가하고 위로 올라가면 y 값이 감소한다. y 값이 음수이면 원점을 넘어선 위치(즉 x축 윗부분)가 된다.

`stopDrag()` 함수를 이용하면 언제든지 드래그를 멈출 수 있다. 한 번에 하나의 무비 클립이나 무비만 드래그할 수 있기 때문에, 새로운 `target`을 이용하여 `startDrag()` 함수를 호출하면 이전에 클립을 드래그하던 작업이 모두 취소된다. 어떤 무비나 무비 클립을 드래그하면 그 안에 포함된 무비 클립도 모두 함께 움직인다.

예제

```
// ball을 드래그한다. 스테이지의 왼쪽 위 구석과 (225, 200)으로 이루어지는
// 직사각형 영역 안에서만 움직일 수 있다.
startDrag("ball", true, 0, 0, 225, 200);
```

참조

`MovieClip.startDrag()`, `stopDrag()`

stop() 전역 함수

현재 프레임에서 무비 재생을 멈추는 메소드

버전 플래시 2 이후

문법 `stop()`

설명

`stop()`은 무비나 무비 클립의 재생을 멈추는 함수로, 매우 간단하지만 플래시에서 꽤 자주 쓰이는 함수이다. `MovieClip.stop()` 메소드와 같은 역할을 하는 전역 함수이며 사용자가 그래픽 메뉴에서 어떤 아이템을 선택할 때까지 기다릴 때 보통 이 함수를 사용한다.

참조

MovieClip.stop(), play()

stopAllSounds() 전역 함수

무비의 사운드를 모두 정지시키는 함수

버전 플래시 3 이후**문법** stopAllSounds()**설명**

stopAllSounds()는 무비에서 현재 재생 중인 사운드를 모두(아무리 여러 단계 아래에 있는 클립이라도 상관없음) 정지시키는 함수이다. 이 함수는 프로그램을 통해 만든 Sound 객체를 포함한 무비의 모든 사운드에 적용된다. 사운드를 재생하고 정지시키고 음량을 조절하는 것과 같이 정밀하게 제어하고 싶다면 Sound 클래스를 참조하기 바란다.

stopAllSounds()는 일시적인 효과만을 나타낸다는 점에 주의하자. stopAllSounds()를 실행한 다음에 다른 사운드를 시작시키면 그 사운드는 다시 재생된다. 영구적으로 무비에서 아무런 소리가 나지 않도록 할 수 있는 방법은 없다.

참조

Sound.setVolume(), Sound.stop()

stopDrag() 전역 함수

진행 중인 드래그 작업을 모두 종료시키는 함수

버전 플래시 4 이후**문법** stopDrag()**설명**

무비 클립이 스테이지 위에서 마우스를 따라 움직이게 할 때는 startDrag() 함수를 사용한다. stopDrag()는 무비 클립이 마우스를 따라 움직이는 작업을 종료시킬 때 사용하는 함수이다. 한 번에 하나의 무비 클립 또는 무비만을 드래그할 수 있기 때문에 stopDrag()를 호출할 때는 target 인자를 사용할 필요가 없다. stopDrag()를 호출하기만 하면 현재 진행 중인 드래그 작업이 종료된다.

온라인 코드 창고의 'Interface Widgets' 부분에 나온 것처럼 `stopDrag()` 함수는 `startDrag()`와 함께 플래시에서 간단한 드래그-앤-드롭 인터페이스를 구현하기 위한 용도로 쓰인다.

예제

아래 버튼 코드를 사용하면 버튼을 누르고 있는 동안에는 무비 클립이 마우스에 끌려다니고 버튼에서 손가락을 떼면 드래그가 중지된다.

```
on (press) {
    startDrag("", true);
}

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

참조

`MovieClip.stopDrag()`, `startDrag()`

String() 전역 함수

주어진 값을 String 데이터형으로 변환하는 함수

버전	플래시 5
문법	<code>String(value)</code>
인자	
<i>value</i>	문자열로 변환할 값을 포함하는 표현식

리턴 값

value를 문자열로 변환한 값

설명

`String()`은 주어진 인자를 문자열 값으로 변환하고 그 값을 리턴하는 함수이다. 다양한 유형의 데이터를 문자열로 변환할 때 적용되는 규칙은 [표 3-2]에 나와 있다. `String()` 함수를 직접 사용해야 하는 경우는 거의 없다. 대부분의 경우에 액션스 크립트에서 필요에 따라 적당히 문자열 형으로 데이터를 변환하기 때문이다.

`String()` 전역 함수와 `String` 클래스 생성자를 혼동하지 않도록 주의하자. `String()` 전역 함수는 주어진 표현식을 문자열로 변환하는 함수이지만, `String` 클래스 생성자는 문자열 데이터를 객체로 감싸서 그 객체의 속성이나 메소드를 사용할 수 있게 해주는 역할을 하기 때문이다.

주의 사항

플래시 4의 .fla 파일을 플래시 5 형식으로 변환하면 `String()` 함수가 종종 사용된다. 플래시 4 파일을 플래시 5 파일로 변환할 때 데이터형이 어떤 식으로 처리되는지 알고 싶다면, 3장의 ‘플래시 4에서 플래시 5로의 데이터형 변환’ 절을 참조하기 바란다.

참조

`String` 클래스, 3장의 ‘직접 형 변환’

String 클래스

문자열 데이터의 래퍼 클래스

버전 플래시 5

생성자 `new String(value)`

인자

value 그 값을 구하여 필요하다면 문자열로 변환한 다음 `String` 객체로 감싸는 표현식

속성

length 문자열에 있는 글자의 개수

클래스 메소드

다음 메소드는 `String` 클래스의 객체가 아닌 `String` 클래스 자체를 통해 호출해야 한다.

`fromCharCode()`

하나 이상의 Latin 1/Shift-JIS 코드 포인트로부터 문자열을 만드는 메소드

메소드

아래 객체 메소드들은 String 클래스의 인스턴스에서 호출해야 한다.

charAt() 문자열의 특정 위치에 있는 글자를 리턴하는 메소드

charCodeAt() 문자열의 특정 위치에 있는 글자의 코드 포인트를 리턴하는 메소드

concat() 하나 이상의 아이템을 하나의 문자열로 합치는 메소드

indexOf() 문자열에서 특정 하위 문자열이 처음으로 나타나는 위치를 찾는 메소드

lastIndexOf()

문자열에서 특정 하위 문자열이 마지막으로 나타나는 위치를 찾는 메소드

slice()

양수 또는 음수로 지정된 글자의 위치를 기준으로 하위 문자열을 추출하는 메소드

split() 문자열을 배열로 변환하는 메소드

substr() 시작 위치와 길이를 기준으로 하위 문자열을 추출하는 메소드

substring() 양수만으로 지정된 글자의 위치를 기준으로 하위 문자열을 추출하는 메소드

toLowerCase()

모든 글자를 소문자로 변환한 문자열을 리턴하는 메소드

toUpperCase()

모든 글자를 대문자로 변환한 문자열을 리턴하는 메소드

설명

String 클래스는 다음과 같은 용도로 쓰인다.

- 이 클래스를 이용하면 문자열의 length 속성을 사용할 수 있으며 indexOf() 나 slice()와 같은 문자열 관련 연산을 처리할 수 있다. 원시 문자열 값에 대해 메소드를 호출하면 인터프리터에서 자동으로 그 문자열로부터 String 객체를 만든다(그리고 작업이 끝나면 그 객체를 제거한다).

- 어떤 유형의 데이터도 문자열로 변환할 수 있다.
- Latin 1이나 Shift-JIS의 코드 포인트를 기준으로 새로운 문자열을 만들기 위해 `fromCharCode()` 클래스 메소드를 호출할 때도 사용된다.
- 원시 문자열 값을 이름이 없는 내부 속성으로 포함하고 있는 String 객체를 만들기 위해 사용할 수도 있다. 하지만 이런 용도로 쓰이는 일은 거의 없다.

주의 사항

String 클래스 생성자는 보통 문자열이 아닌 데이터를 문자열로 변환할 때 주로 쓰인다. 자세한 내용은 `String()` 전역 함수 부분을 참조하기 바란다.

참조

‘4장. 원시 데이터형’의 ‘문자열’

String.charAt() 메소드

문자열의 특정 위치에 있는 글자를 리턴하는 메소드

버전 플래시 5

문법 `string.charAt(index)`

인자

index 가져올 문자의 위치를 나타내는 정수. 0(첫 글자) 이상, `string.length - 1`(마지막 글자) 이하의 값을 사용해야 한다.

리턴 값

`string`에서 `index` 위치에 있는 글자

설명

`charAt()`은 문자열에서 특정 위치(`index`)에 있는 글자를 리턴하는 메소드이다.

예제

```
trace("It is 10:34 pm".charAt(1)); // "t"가 출력된다(두 번째 글자).
var country = "Canada";
trace(country.charAt(0));          // "C"가 출력된다(첫 글자).
```

```
// 이 함수에서는 문자열에 있는 모든 스페이스를 없애고 그 결과를 리턴한다.
function stripSpaces (inString) {
  var outString = "";
  for (i = 0; i < inString.length; i++) {
    if (inString.charAt(i) != " ") {
      outString += inString.charAt(i);
    }
  }
  return outString;
}
```

참조

String.charCodeAt(), String.indexOf(), String.slice(); 4장의 'charAt() 함수'

String.charCodeAt() 메소드

문자열의 특정 위치에 있는 글자의 코드 포인트를
구하는 메소드

버전 플래시 5

문법 string.charCodeAt(index)

인자

index 문자열에 있는 글자의 위치를 나타내는 정수. 0(첫 글자) 이상 string.length-1(마지막 글자) 이하의 값을 사용해야 한다.

리턴 값

string의 index 위치에 있는 글자의 Latin 1이나 Shift-JIS 코드 포인트(부록 B 참조)를 나타내는 정수

예제

```
var msg = "A is the first letter of the Latin alphabet.";
trace(msg.charCodeAt(0)); // 65가 출력된다("A"의 코드).
trace(msg.charCodeAt(1)); // 32가 출력된다(공백 문자의 코드).
```

참조

String.charAt(), String.fromCharCode(); 부록 B, 4장의 'charCodeAt() 함수'

String.concat() 메소드

하나 이상의 값을 하나의 문자열로 합치는 메소드

버전 플래시 5**문법** `string.concat(value1, value2, ...valuen)`**인자***value1...valuen*

문자열로 변환하여 string과 합칠 값

리턴 값

string에 value1, value2, ...valuen을 합친 문자열

설명

concat() 메소드는 일련의 값에서 문자열을 만든다. 합치기 연산자(+)를 이용하는 것과 똑같지만 + 연산자는 덧셈 연산자로도 쓰이기 때문에, 혼란을 피하기 위해 concat() 메소드를 사용하는 경우도 종종 있다. concat()에 대한 자세한 내용은 4장을 참조하기 바란다.

주의 사항

concat()을 실행시켜도 string 변수의 내용은 바뀌지 않고 새로운 문자열로 결과를 리턴한다는 점에 주의하자.

예제

```
var greeting = "Hello";
excitedGreeting = greeting.concat("!");
trace(greeting);           // "Hello"가 출력된다.
trace(excitedGreeting);    // "Hello!"가 출력된다.

var x = 4;                  // x를 정수로 초기화한다.
trace(x + 5);              // 9가 출력된다.
trace(x.concat(5));        // x가 문자열이 아니므로 작동하지 않는다.
trace(String(x).concat(5)); // "45"가 출력된다.

var x = "4";               // x를 문자열로 초기화한다.
trace(x.concat(5));        // "45"가 출력된다.
trace(concat("foo", "fee")); // concat()은 메소드 형태로 호출해야
                             // 하므로 이런 식으로 사용할 수 없다.
```


참조

5장의 + 연산자, 4장의 'concat() 함수'

String.fromCharCode() 클래스 메소드

하나 이상의 코드 포인트를 이용하여
문자열을 만드는 메소드

버전 플래시 5

문법 `String.fromCharCode(code_point1, code_point2,
...code_pointn)`

인자

code_point1,...code_pointn

Latin 1 또는 Shift-JIS 문자 코드 포인트('부록 B. Latin 1 문자 범위 및
키코드' 참조)에 해당하는 한 개 이상의 십진 정수

리턴 값

주어진 코드 포인트로 표현되는 글자들을 합쳐서 만든 문자열

설명

fromCharCode()는 4장에서 설명한 것처럼 코드 포인트를 이용하여 하나의 글
자 혹은 여러 개의 글자들을 만드는 클래스 메소드이다.

예제

```
// copyright 기호 뒤에 2001을 덧붙인 문자열을 만든다.  
copyNotice = String.fromCharCode(169) + "2001";
```

참조

String.charCodeAt(); 부록 B, 4장의 'fromCharCode() 함수'

String.indexOf() 메소드

문자열에서 특정 하위 문자열이 처음 나타나는 위치를
구하는 메소드

버전 플래시 5

문법 `string.indexOf(substring)`
`string.indexOf(substring, startIndex)`

인자

substring 검색할 내용이 들어있는 문자열

startIndex substring 검색을 시작할 위치를 나타내는 정수. 0(첫 글자) 이상 `string.length-1`(마지막 글자) 이하의 정수를 사용해야 한다. 기본값은 0이다(옵션).

리턴 값

`startIndex`에서 시작하여 `string`에서 `substring`이 나타나는 첫 번째 위치. `string`의 `startIndex` 이후 부분에서 `substring`을 찾지 못하면 -1을 리턴한다.

설명

`indexOf()`는 문자열에서 특정 내용을 검색하거나 문자열에 특정 내용이 들어있는지 알아보기 위한 용도로 쓰이는 메소드이다.

예제

```
// 이메일 주소에 @ 기호가 들어있는지 확인한다.
var email = "derekaol.com";
if (email.indexOf("@") == -1) {
    trace ("This isn't a valid email address");
}

// 이메일 주소에 @ 기호가 들어있는지 확인하고
// aol.com 도메인에 속하는지 확인한다.
var email = "derek@aol.com";
var atPos = email.indexOf("@");
if (atPos != -1 && email.indexOf("aol.com") == atPos + 1) {
    gotoAndStop("AOLuserOffer");
}
```

아래 예제는 단답형 주관식 퀴즈의 점수를 매길 때 사용할 수 있는 문자열에서 특정 키워드를 찾을 때 쓸 수 있는 코드이다.

```
// 대소문자를 구분하지 않고 origStr에서 searchStr을
// 검색하는 함수
function search (origStr, searchStr) {
  var origStr = origStr.toLowerCase();
  var searchStr = searchStr.toLowerCase();
  return origStr.indexOf(searchStr) != -1;
}

var answer = "einstein";
var guess = "Dr. Albert Einstein";

// guess에 "einstein"이 있으면 score를 1 증가시킨다.
if (search(guess, answer)) {
  score++;
}
```

참조

String.charAt(), String.lastIndexOf(); 4장의 'indexOf() 함수'

String.lastIndexOf() 메소드

문자열에서 특정 하위 문자열이 처음 나타나는 위치를
구하는 메소드

버전 플래시 5

문법 string.lastIndexOf(substring)
string.lastIndexOf(substring, startIndex)

인자

substring 검색할 내용을 나타내는 문자열

startIndex substring 검색을 시작할 위치를 나타내는 정수. string을 검색할 때
startIndex 위치에서 시작하여 거꾸로 검색하므로 0(첫 글자) 이상
string.length-1(마지막 글자) 이하의 정수를 사용해야 한다. 기본값은
string.length-1이다(옵션).

리턴 값

string의 startIndex 이전 위치에서 substring이 마지막으로 나오는 위치. string의 startIndex 앞부분에 substring이 없으면 -1이 리턴된다.

설명

lastIndexOf()는 어떤 문자열에서 주어진 하위 문자열이 마지막으로 나타나는 위치를 찾아내거나 문자열에 특정 하위 문자열이 들어있는지 확인하기 위한 용도로 쓰이는 메소드이다.

예제

```
URL = "http://www.moock.org/webdesign/flash/llthethewindow.html";
// 마지막 슬래시 문자를 찾아낸다.
lastSlash = URL.lastIndexOf("/");
// URL에서 파일 이름을 뽑아낸다.
file = URL.substring(lastSlash + 1);
trace(file); // llthethewindow.html이 출력된다.
```

참조

String.charAt(), String.indexOf(); 4장의 'lastIndexOf() 함수'

String.length 속성

문자열에 포함된 글자의 개수

버전 플래시 5

문법 *string.length*

액세스 읽기 전용

설명

length 속성은 string에 있는 글자의 개수를 나타낸다. 다른 몇몇 언어에서와 같이 널 문자(ASCII 0)로 문자열의 끝을 표시하기 않기 때문에, length 값을 구할 때 널 문자는 제외시킨다. 예를 들면 다음과 같다.

```
// 문자열 "A" + null + "B"를 만든다.
var myString = String.fromCharCode(65,0,66);
trace(myString.length); // 2가 출력된다(널 문자는 무시됨).
```

예제

```

var myString = "hello";
trace (myString.length); // 5가 출력됨
trace ("hello".length); // 5가 출력됨

// 숫자 1000을 문자열로 변환하여
// 그 길이를 테스트해보자.
var age = 1000;
// 숫자의 자리수가 틀리면 오류 메시지를 출력하는 코드
if (String(age).length != 2) {
    trace ("Please enter a two-digit number");
}

```

참조

Array.length(), 4장의 'length 속성'

String.slice() 메소드

양수 또는 음수 인덱스를 기준으로 문자열에서 하위
문자열을 뽑아내는 메소드

버전 플래시 5

문법 *string.slice(startIndex, endIndex)*

인자

startIndex string에서 뽑아낼 첫 번째 글자의 위치를 나타내는 정수. startIndex가 음수이면 문자열의 맨 뒤에서부터 위치를 계산한다(-1은 마지막 글자, -2는 마지막에서 두 번째 글자..., startIndex를 음수로 지정하면 string.length+startIndex 위치를 가리키게 된다).

endIndex string에서 뽑아낼 마지막 글자 다음 글자의 위치를 나타내는 정수. endIndex가 음수이면 문자열의 맨 뒤에서부터 위치를 계산한다(-1은 마지막 글자, -2는 마지막에서 두 번째 글자..., 즉 endIndex를 음수로 지정하면 string.length+endIndex 위치를 가리키게 된다). 이 값을 지정하지 않으면 string.length가 기본값으로 쓰인다(옵션).

리턴 값

startIndex에서 시작해서 endIndex-1에서 끝나는 string의 하위 문자열 (substring). startIndex와 endIndex는 0을 기준으로 하는 인덱스이다.

설명

slice() 메소드는 문자열에서 하위 문자열을 추출하는 데 쓰이는 세 가지 메소드 (나머지 둘은 substring()과 substr()) 중 하나이다. slice() 메소드를 이용하면 시작과 끝을 나타내는 인덱스를 음수로 지정할 수 있으며, 이 기능을 이용하여 문자열의 맨 뒤부터 뽑아낼 하위 문자열의 위치를 계산할 수 있다.

주의 사항

slice() 메소드에서는 string을 변경시키지 않고 새로운 문자열로 결과를 리턴한다.

예제

```
var fullName = "Steven Sid Mumby";
middleName = fullName.slice(7, 10); // middleName에 "Sid"를 대입한다.
middleName = fullName.slice(-9, -6); // middleName에 "Sid"를 대입한다.
```

참조

String.substr(), String.substring(); 4장의 'slice() 함수'와 '문자 검색과 문자 추출 결합'

String.split() 메소드

문자열을 일련의 배열 요소로 변환하는 메소드

버전 플래시 5

문법 *string.split(delimiter)*

인자

delimiter 새로운 배열의 원소를 만들 때 string에서 각 원소를 구분하는 기준이 되는 한 개 이상의 문자

리턴 값

string을 delimiter를 기준으로 쪼개어 만든 하위 문자열을 포함하는 배열

설명

`split()`은 문자열을 하위 문자열로 나누고 그러한 하위 문자열을 배열의 원소에 집어넣고 그 배열을 리턴하는 메소드이다. `delimiter`가 연속으로 나오는 경우에는 비어있는 원소가 만들어진다. 예를 들어 다음과 같은 코드를 사용하면

```
owners = "terry,doug,,,jon";
ownersArray = owners.split(",");
```

`ownersArray`의 원소는 다음과 같이 구성된다(2번과 3번 원소는 `undefined`).

```
0: terry
1: doug
2:
3:
4: jon
```

`split()` 메소드는 주로 CGI 스크립트나 텍스트 파일에서 가져온 문자열을 조작하기 위해 배열에 저장하는 용도로 쓰인다. 함수에 하나의 문자열 인자만 전달할 수 있는 HTML 텍스트 필드 <A> 태그에서 쓰이는 `asfunction` 호출의 매개변수를 파싱할 때도 이 메소드를 많이 사용한다(18장의 'HTML 링크에서 액션스크립트 함수 호출하기' 부분 참조). 구분자(`delimiter`)로는 보통 탭 문자나 쉼표를 많이 사용한다.

예제

`names.txt`라는 텍스트 파일에 이름 목록을 저장하는 경우를 생각해 보자. 아래 나온 것처럼 각 이름은 탭 문자로 구분한다.

```
owners=terry    doug    jon
```

무비의 1번 프레임에서 `names.txt`를 무비로 불러오자.

```
this.loadVariables("names.txt");
```

`names.txt`를 완전히 로딩한 다음('10장. 이벤트 및 이벤트 핸들러'의 '데이터' 부분 참조), `owners` 변수에 저장된 내용으로부터 배열을 만든다.

```
splitString = String.fromCharCode(9); // spliteString에 탭 문자를 대입한다.
ownersArray = owners.split(splitString);
```

```
trace(ownersArray[1]); // "doug"가 출력된다.
```

텍스트 크기가 커지면 `split()`을 실행하는 데 걸리는 시간이 길어질 수 있다는 점에 주의하자. 실행 속도가 빨라야 한다면 데이터를 적당한 크기로 분리하거나 XML을 사용하는 것을 고려해보는 것도 괜찮다. XML 클래스 부분을 참조해 보자.

버그

플래시 5에서 비어있는 문자열을 구분자로 지정하면, `string` 전체가 새로 만드는 배열의 첫 번째 원소로 들어간다. ECMA-262에 의하면 비어있는 문자열을 구분자로 사용하면, `string`의 각 글자가 하나씩 배열의 원소로 저장된다. 이와 마찬가지로 플래시 5에서는 구분자가 여러 글자이면 `string` 전체가 배열의 첫 번째 원소로 들어가게 된다.

참조

`Array.join()`, 4장의 ‘`splice()` 함수’

String.substr() 메소드

시작 위치와 길이를 기준으로 문자열에서 하위 문자열을 뽑아내는 메소드

버전 플래시 5

문법 `string.substr(startIndex, length)`

인자

startIndex `string`에서 뽑아낼 첫 글자의 위치를 나타내는 정수. `startIndex`가 음수이면 문자열의 끝부터 위치를 계산한다(-1은 마지막 글자, -2는 마지막에서 두 번째 글자..., 즉 `startIndex` 값을 음수로 지정하면 `string.length+startIndex`가 첫 글자의 위치가 된다).

length `startIndex`부터 뽑아낼 글자의 개수. 이 값을 지정하지 않으면 `startIndex` 위치부터 `string` 끝까지 모두 뽑아낸다(옵션).

리턴 값

`startIndex`에서 시작하여 `length` 개의 글자가 들어있는 `string`의 하위 문자열. `length` 값을 생략하면 `startIndex`부터 `string` 끝까지 모두 뽑아낸다.

설명

substr() 메소드는 문자열에서 하위 문자열을 뽑아내는 세 가지 메소드(나머지는 slice()와 substring()) 중 하나이다. substr()에서는 두 개의 인덱스가 아닌 하나의 인덱스와 하위 문자열의 길이를 기준으로 문자열을 뽑아낸다.

주의 사항

substr() 메소드를 실행시켜도 string 자체는 변경되지 않고 실행 결과를 새로운 문자열로 리턴한다.

예제

```
var fullName = "Steven Sid Mumby";

middleName = fullName.substr(7, 3); // middleName에 "Sid"를 대입한다.
firstName  = fullName.substr(0, 6); // firstName에 "Steven"을 대입한다.
lastName   = fullName.substr(11);  // lastName에 "Mumby"를 대입한다.

// 시작 위치를 음수로 지정하는 방법을 살펴보자.
middleName = fullName.substr(-9, 3); // middleName에 "Sid"를 대입한다.
firstName  = fullName.substr(-16, 6); // firstName에 "Steven"을 대입한다.
lastName   = fullName.substr(-5);    // lastName에 "Mumby"를 대입한다.
```

참조

String.slice(), String.substring(), 4장의 'substr() 함수'와 '문자 검색과 문자 추출 결합'

String.substring() 메소드

양수로 나타낸 글자 위치를 기준으로 문자열에서 하위 문자열을 뽑아내는 메소드

버전 플래시 5

문법 string.substring(startIndex, endIndex)

인자

startIndex string에서 뽑아낼 첫 글자의 위치를 나타내는 양의 정수. 이 값이 음수이면 0부터 시작한다.

endIndex string에서 뽑아낼 마지막 글자 바로 뒤에 있는 글자의 위치를 나타내는 양의 정수. 이 값을 생략하면 string.length가 기본값으로 쓰인다. 음수이면 0이 대신 쓰인다.

리턴 값

startIndex에서 시작해서 endIndex-1로 끝나는 string의 하위 문자열. startIndex와 endIndex는 모두 0을 기준으로 하는 인덱스이다.

설명

substring() 메소드는 문자열에서 하위 문자열을 추출하는 데 쓰이는 세 가지 메소드(나머지는 slice()와 substr()) 중 하나이다. substring() 함수는 slice()와 똑같지만 인자에 음수를 사용할 수 없고 endIndex가 startIndex보다 크면 자동으로 둘의 순서를 바꿔준다는 점에서 차이가 난다.

주의 사항

substring()을 실행시켜도 string 자체는 바뀌지 않는다. 실행 결과는 새로운 문자열로 리턴된다.

예제

```
// 문자열에서 이름을 추출한다.
var fullName = "Steven Sid Mumby";
middleName = fullName.substring(7, 10); // middleName에 "Sid"를
// 대입한다.

middleName = fullName.substring(10, 7); // middleName에 "Sid"를
// 대입한다.
// (인덱스가 자동으로 바뀐다)

firstName = fullName.substring(0, 6); // firstName에 "Steven"을
// 대입한다.

lastName = fullName.substring(11); // lastName에 "Mumby"를
// 대입한다.
```

아래 코드는 문자열에서 하위 문자열을 찾아서 다른 문자열로 바꾸는 기능을 구현하는 함수의 예이다.

```
// 찾아 바꾸기 함수
function replace (origStr, searchStr, replaceStr) {
    var tempStr = "";
```

```
var startIndex = 0;
if (searchStr == "") {
    return origStr;
}

if (origStr.indexOf(searchStr) != -1) {
    while ((searchIndex = origStr.indexOf(searchStr, startIndex)) != -
1) {
        tempStr += origStr.substring(startIndex, searchIndex);
        tempStr += replaceStr;
        startIndex = searchIndex + searchStr.length;
    }
    return tempStr + origStr.substring(startIndex);
} else {
    return origStr;
}
}

msg = "three times three is four";
trace(replace(msg, "three", "two")); // "two times two is four"
// 라고 출력된다.
```

참조

String.slice(), String.substr(); 4장의 ‘문자 검색과 문자 추출 결합’ 과 ‘substring() 함수’

String.toLowerCase() 메소드

문자열에 있는 대문자를 모두 소문자로
바꾼 문자열을 리턴하는 메소드

버전 플래시 5

문법 *string*.toLowerCase()

리턴 값

string에 있는 모든 대문자를 소문자로 바꾼 문자열. 대문자가 없는 글자들은 바뀌지 않는다.

설명

`toLowerCase()` 메소드에서는 `string`의 대문자를 모두 소문자로 바꾼 새로운 문자열을 만든다. 소문자만 사용해야 하는 경우에 모든 대문자를 소문자로 변환하거나 대소문자를 구분하지 않고 문자열을 비교해야 하는 경우에 이 메소드를 사용하면 된다. `toLowerCase()` 메소드에서는 A-Z 범위에 있는 글자만 변환한다(엑센트나 움라우트가 있는 글자는 건드리지 않는다).

주의 사항

`toLowerCase()`에서는 `string` 자체를 바꾸지 않는다. 결과는 새로운 문자열로 리턴된다.

예제

```
// msg에 "this sentence has mixed caps!"가 대입된다.
msg = "ThiS SenTencE Has MixED CaPs!".toLowerCase();

// 두 문자열을 비교한다(대소문자 구분하지 않음).
function caseInsensitiveCompare (stringA, stringB) {
    return (stringA.toLowerCase() == stringB.toLowerCase());
}

trace(caseInsensitiveCompare("Colin", "colin")); // true가 출력된다.
```

참조

`String.toUpperCase()`; 4장의 'toLowerCase() 함수'

String.toUpperCase() 메소드

문자열에 있는 모든 소문자를 대문자로 바꾼 문자열을 리턴하는 메소드

버전 플래시 5

문법 `string.toUpperCase()`

리턴 값

`string`에 있는 모든 소문자를 대문자로 바꾼 문자열. 소문자가 없는 글자들은 바뀌지 않는다.

설명

`toUpperCase()` 메소드에서는 `string`의 소문자를 모두 대문자로 바꾼 새로운 문자열을 만든다. 대문자만 사용해야 하는 경우에 모든 소문자를 대문자로 변환하거나 대소문자를 구분하지 않고 문자열을 비교해야 할 때 이 메소드를 사용하면 된다. `toUpperCase()` 메소드에서는 `a-z` 범위에 있는 글자만 변환한다(엑센트나 움라우트가 있는 글자는 건드리지 않는다).

주의 사항

`toUpperCase()`에서는 `string` 자체를 바꾸지 않는다. 결과는 새로운 문자열로 리턴된다.

예제

```
"listen to me".toUpperCase(); // "LISTEN TO ME"가 리턴된다.  
var msg1 = "Your Final Score: 234";  
var msg2 = msg1.toUpperCase(); // msg2에 "YOUR FINAL SCORE: 234"가 대입된다.
```

참조

`String.toLowerCase()`; 4장의 ‘`toUpperCase()` 메소드’

targetPath() 전역 함수

무비 또는 무비 클립에 대한 절대 경로를 리턴하는 함수

버전 플래시 5

문법 `targetPath(movieClip)`

인자

movieClip 무비 클립 객체에 대한 레퍼런스

리턴 값

`movieClip`의 절대 경로를 점 표기법으로 나타낸 문자열(예: “_level0.myMovie”)

설명

`targetPath()` 함수는 무비 클립의 레퍼런스를 클립에 대한 절대 경로로 나타내는 문자열(`MovieClip.valueOf()`의 리턴 값과 같음)을 리턴한다. `targetPath()` 함수는 그 클립이 속해있는 타임라인에 따라 다르게 작동하는 코드를 만들 때 종종 쓰인다.

버그

플래시 5 액션스크립트 디렉터리의 `targetPath()` 함수를 설명하는 부분에 나온 예제는 `targetPath()`의 적절한 사용법을 보여주지 못한다. 그 예제에 나온 것과는 달리 `targetPath()` 함수는 `tellTarget()` 함수와 같은 역할을 하지 않는다.

예제

`square`라는 무비 클립이 메인 0번 레벨의 메인 타임라인에 있는 `shapes`라는 클립에 포함된다면 `shapes` 클립에서 다음과 같은 선언문을 실행했을 때

```
targetPath(square);
```

다음과 같은 결과가 리턴된다.

```
"_level0.shapes.square"
```

참조

`MovieClip._target`, `MovieClip.valueOf()`; 13장의 'targetPath() 함수'

tellTarget() 전역 함수

선언문을 원격 무비 클립의 영역에서 실행시키는 함수

버전

플래시 3 및 플래시 4(플래시 5에서는 더 이상 쓰이지 않고 객체 지향적인 방법을 사용하거나 `with` 선언문을 사용하는 것이 좋다)

문법

```
tellTarget (target) {  
    statements  
}
```

인자

target 무비나 무비 클립 인스턴스에 대한 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 레퍼런스를 사용하면 자동으로 경로를 나타내는 문자열로 변환된다)

statements target의 영역에서 실행시킬 선언문

설명

플레이시 3와 플레이시 4에서는 두 개의 무비 클립 사이에서 의사소통(한 무비 클립에서 다른 무비 클립을 제어하는 것)할 때 `tellTarget()`을 이용하는 방법을 가장 많이 사용한다. 원격 무비 클립에서 `play()`, `stop()`, `gotoAndStop()`과 같은 함수를 실행시킬 때 `tellTarget()`을 이용한다. 액션스크립트에 변수 기능이 추가된 플레이시 4에서는 `tellTarget()`을 이용하여 원격 클립 변수 값을 구하거나 그 변수에 새로운 값을 대입할 수 있다. 하지만 플레이시 5에서 이러한 작업은 점 연산자(`.`)나 배열 액세스 연산자(`[]`)를 이용하여 처리할 수 있다. 또한 `tellTarget()` 함수 대신 '6장. 선언문'에 나온 `with` 선언문을 사용할 수도 있다.

주의 사항

`tellTarget()` 함수를 사용할 때는 선언문 블록을 사용해야 하므로 이 함수는 사실 함수보다는 선언문에 더 가깝다. 하지만 `tellTarget()`은 더 이상 쓰이지 않으므로 그다지 신경쓰지 않아도 된다.

예제

```
tellTarget ("ball") {
    gotoAndStop("redStripes");
    _x += 300;
}
```

참조

6장의 'with 선언문', 13장의 'Tell Target은 어디로?' 와 '무비 클립의 객체성'

toggleHighQuality() 전역 함수

플레이어의 렌더링 화질을 바꾸는 함수

버전 플래시 2 이후(플래시 5에서는 이 함수 대신 `_quality` 전역 속성을 사용한다)

문법 `toggleHighQuality()`

설명

이 함수는 렌더링 화질을 고화질 또는 저화질로 바꿔준다. 렌더링 화질이 고화질이면 플래시 플레이어에서는 선을 렌더링할 때 경계를 안티앨리어싱 처리(부드럽게 표시됨)한다. 저화질로 설정해 놓으면 선을 앨리어싱 처리(계단 모양으로 표시됨)한다. `toggleHighQuality()` 함수에는 인자를 사용하지 않아도 된다. 매번 호출할 때마다 High에서 Low로, Low에서 High로 설정이 바뀐다. 하지만 이러한 방법은 어떤 화질이 설정되어 있는지 정확히 알 수도 없을 뿐더러 화질 설정을 두 가지밖에 할 수 없다는 단점이 있다.

플래시 5부터 `toggleHighQuality()`는 더 이상 쓰이지 않으며, 대신 Low, Medium, High, Best 렌더링 설정을 지원하는 `_quality` 전역 속성을 사용한다.

참조

`_highquality`, `_quality`

trace() 전역 함수

주어진 값을 Output 창에 표시하는 함수

버전 플래시 4 이후

문법 `trace(value)`

인자

value Output 창으로 출력할 표현식. `value`를 계산한 값이 문자열이 아니면 Output 창에 표시하기 전에 [표 3-2]에 나온 규칙에 따라 문자열로 변환된다.

설명

`trace()`는 플래시 제작 환경의 Test Movie 모드에서만 사용할 수 있는 디버깅 도구로 쓰이는 함수이다. 조금 이상해 보일지 모르지만 `trace()` 함수는 사실 액션스크립트 프로그래밍에서 매우 기본적인 구성요소 중 하나이다. 이 함수를 이용하면 무비를 재생하는 도중에 아무 때나 변수나 표현식 값을 확인할 수 있기 때문이다.

주의 사항

`trace()`를 사용하면 무비 재생 속도가 많이 느려진다. File → Publish Settings → Flash에서 Omit Trace Actions 옵션을 선택하여 `trace()` 함수가 실행되지 않도록 설정할 수 있다.

예제

```
trace(firstName);           // firstName의 값을 출력한다.
trace(myClip);              // myClip의 경로를 출력한다.
trace(myClip._x)            // myClip의 x 좌표를 출력한다.
trace("hello" + "there");   // 표현식을 계산하고 그 결과를 출력한다.
```

참조

‘19장. 디버깅’

unescape() 전역 함수

이스케이프 문자열을 디코딩하는 함수

버전 플래시 5

문법 `unescape(stringExpression)`

인자

stringExpression

`escape()`를 이용하여 인코딩한 문자열(또는 계산 결과가 문자열인 표현식)

리턴 값

`stringExpression`을 디코딩한 새로운 문자열

설명

unescape()는 stringExpression으로부터 만든 새로운 문자열을 리턴하는 함수이다. 리턴된 문자열에는 stringExpression에서 % 기호 뒤에 나오는 두 자리 16진수로 이스케이프 처리된 문자들을 Latin 1 문자로 변환한 문자열이 들어있다. escape()와 unescape() 함수는 네트워크를 통해 전송하기 위해 문자열을 인코딩하고 디코딩하기 위한 용도로 쓰이는 함수이다. 하지만 플래시에서 URL 인코딩된 텍스트를 loadVariables()를 이용하여 가져올 때는 자동으로 원래 문자열로 변환되기 때문에 직접 unescape() 함수를 사용하는 경우는 거의 없다.

예제

```
var msg = "hello!";
// msgCoded를 "hello%21"로 설정한다.
msgCoded = escape(msg);
// msgCode를 디코딩하여 "hello!"로 바꾼다.
var msgDecoded = unescape(msgCoded);
```

참조

escape(); ‘부록 B. Latin 1 문자 범주 및 키코드’

unloadMovie() 전역 함수

플레이어에서 무비 또는 무비 클립을 제거하는 함수

버전 플래시 4 이후(플래시 5의 unloadMovie()는 플래시 4의 Unload Movie에서 대상 경로를 이용한 것에 해당한다)

문법 unloadMovie(target)

인자

target 플레이어에서 제거할 문서 또는 무비 클립의 경로를 나타내는 문자열 또는 레퍼런스(문자열이 들어갈 자리에 레퍼런스를 사용하면 경로로 자동 변환된다)

설명

unloadMovie()는 플레이어의 문서 레벨에서 무비를 제거하는 데 쓰이는 함수이다. 예를 들어 제거하고자 하는 무비가 플레이어의 1번 레벨에 있다면 다음과 같은 명령을 사용하면 된다.

```
unloadMovie("_level1");
```

unloadMovie() 함수는 무비 클립 인스턴스를 지우기 위한 용도로 사용할 수 있으며, 이 때 인스턴스에 있는 내용만 삭제할 뿐 인스턴스 자체를 없애지는 않는다. 인스턴스는 스테이지에 비어있는 껍데기처럼 남게 되며 그 인스턴스에 다른 무비를 불러올 수 있다. 따라서 loadMovie()와 unloadMovie()를 이용하면 하나의 클립을 여러 가지 다양한 내용을 담을 수 있는 컨테이너로 활용할 수 있다.

참조

loadMovie(), MovieClip.unloadMovie(), unloadMovieNum(), 13장의 '메인 무비 및 클립 인스턴스 제거', '메소드와 전역 함수가 겹치는 문제'

unloadMovieNum() 전역 함수

특정 문서 레벨의 무비를 제거하는 함수

버전 플래시 3 이후(플래시 5의 unloadMovieNum() 번호가 붙어있는 레벨에 대해서만 사용할 수 있는 플래시 3의 unload Movie 액션에 해당한다)

문법 unloadMovieNum(level)

인자

level 제거할 무비가 들어있는 문서 레벨을 나타내는 음이 아닌 정수 또는 음이 아닌 정수값을 가지는 표현식

설명

unloadMovieNum()은 무비를 제거할 레벨을 문자열이 아닌 레벨 번호를 나타내는 숫자로 표시한다는 점을 제외하면 unloadMovie()와 거의 똑같다. unloadMovieNum()에서는 문서 레벨에 있는 무비만을 제거할 수 있으며 무비 클립은 없앨 수 없다. 일반적으로 제거할 무비의 레벨을 동적으로 할당할 때 다음과 같은 방식으로 사용한다.

```
var x = 3;
unloadMovieNum(x);
```

하지만 `unloadMovie()` 함수를 다음과 같은 식으로 호출하면 `unloadMovieNum()` 과 같은 효과를 나타낼 수 있다.

```
unloadMovie("_level" + x);
```

참조

`loadMovieNum()`, `MovieClip.unloadMovie()`, `unloadMovie()`; 13장의 '메인 무비 및 클립 인스턴스 제거'

updateAfterEvent() 전역 함수

프레임 중간에 스테이지 내용을 렌더링하는 함수

버전 플래시 5

문법 `updateAfterEvent()`

설명

플래시 플레이어에서 프레임을 렌더링하는 사이에 사용자 입력 클립 이벤트 핸들러(`mouseMove`, `mouseDown`, `mouseUp`, `keyDown`, `keyUp`)가 실행되는 경우가 종종 있다. 사용자 입력 클립 이벤트 핸들러에 의해 화면의 내용이 바뀔 경우에 그 변경 사항을 바로 반영하고 싶다면 핸들러 내부에서 `updateAfterEvent()` 함수를 호출하면 된다. 하지만 `updateAfterEvent()` 함수는 아무 때나 화면을 갱신하기 위한 함수가 아니라는 점에 주의하자. 이 함수는 사용자 입력 클립 이벤트 핸들러 안에서만 사용할 수 있다. `onClipEvent()` 핸들러 외부에서는 이 함수를 사용해도 화면이 갱신되지 않는다.

예제

다음과 같은 스크립트를 무비 클립에 추가하면 클립이 마우스 포인터를 움직이고, 마우스가 움직일 때마다 화면이 갱신된다. 포인터가 움직일 때마다 화면을 갱신하기 때문에 클립이 매우 부드럽게 움직인다.

```
onClipEvent (mouseMove) {  
    _x = _root._xmouse;  
    _y = _root._ymouse;  
    updateAfterEvent();  
}
```

버그

플래시 5 액션스크립트 디렉터리에는 `updateAfterEvent()` 함수에서 클립 이벤트를 인자로 받아들이는다고 나와 있지만 실제 이 함수에서는 아무런 인자를 받아들이지 않는다.

참조

10장의 ‘`updateAfterEvent`를 이용한 화면 갱신’

\$version 전역 속성

플래시 플레이어의 버전

버전 플래시 4r11 이후(플래시 5에서는 `getVersion()` 함수를 대신 사용한다)

문법 `_root.$version`

액세스 읽기 전용

설명

`$version` 속성에는 `getVersion()` 전역 함수에서 리턴하는 것과 같은 문자열이 저장된다(운영 체제, 플레이어 버전 정보). `$version` 속성은 플래시 4 플레이어 중간에 도입되었지만 플래시 5에서는 `getVersion()` 함수로 대체되었다. 정확하게 말하자면 전역 속성이 아니라 메인 무비 타임라인의 속성이다. 따라서 다른 무비 클립 타임라인에서 사용할 때는 반드시 `_root.$version` 과 같은 식으로 표기해야 한다.

참조

`getVersion()`

XML 클래스

XML 구조 데이터를 DOM 기반으로 지원하는 클래스

버전	플래시 5
생성자	<code>new XML()</code> <code>new XML(source)</code>
인자	
<i>source</i>	XML 객체 계층으로 파싱할 XML(또는 HTML) 데이터가 저장된 문자열(옵션)
속성	
<i>attributes</i>	원소 특성을 속성 형태로 저장하기 위한 객체
<i>childNodes</i>	노드의 자식에 대한 레퍼런스 배열
<i>contentType</i>	서버로 전송될 MIME 유형
<i>docTypeDecl</i>	문서의 DOCTYPE 태그
<i>firstChild</i>	노드의 첫 번째 자식에 대한 레퍼런스
<i>ignoreWhite</i>	XML을 파싱할 때 공백 노드를 무시할지를 나타내는 속성
<i>lastChild</i>	노드의 마지막 자식에 대한 레퍼런스
<i>loaded</i>	<code>load()</code> 또는 <code>sendAndLoad()</code> 작업의 상태
<i>nextSibling</i>	객체 계층에서 같은 레벨에 있는 바로 다음 노드에 대한 레퍼런스
<i>nodeName</i>	현재 노드의 이름
<i>nodeType</i>	현재 노드의 유형
<i>nodeValue</i>	현재 노드의 값
<i>parentNode</i>	현재 노드의 부모 노드에 대한 레퍼런스
<i>previousSibling</i>	객체 계층에서 같은 레벨에 있는 바로 앞 노드에 대한 레퍼런스

status XML 소스를 객체 계층으로 파싱한 결과를 나타내는 오류 코드

xmlDecl 문서의 XML 선언 태그

메소드

appendChild()

노드에 새로운 자식 노드를 추가하는 메소드

cloneNode() 어떤 노드의 복사본을 만드는 메소드

createElement()

새로운 원소 노드를 만드는 메소드

createTextNode()

새로운 텍스트 노드를 만드는 메소드

hasChildNodes()

자식 노드가 있는지 확인하는 메소드

insertBefore()

노드 앞에 형제(sibling) 노드를 추가하는 메소드

load() 외부 문서에서 XML 소스를 불러오는 메소드

parseXML() XML 소스 코드로 이루어진 문자열을 파싱하는 메소드

removeNode()

객체 계층에서 노드를 제거하는 메소드

send() XML 소스를 스크립트로 보내는 메소드

sendAndLoad()

XML 소스를 스크립트로 보내고 그 소스에서 리턴되는 XML 코드를 받아들이는 메소드

toString() XML 객체를 문자열로 변환하는 메소드

이벤트 핸들러

onData() 외부 XML 소스를 불러오는 작업이 완료되었을 때 실행되는 핸들러

onLoad() 외부 XML 데이터를 객체 계층으로 파싱하는 작업이 완료되었을 때 실행되는 핸들러

설명

XML 클래스의 객체는 플래시에서 XML(또는 HTML) 문서의 내용을 객체지향적인 방법으로 조작하거나 XML 형식의 데이터를 주고받는 데 쓰인다. XML 객체의 메소드와 속성을 이용하면 XML 구조를 갖춘 문서를 만들거나 읽어들이 수 있고 그 문서에 있는 정보를 효율적으로 액세스하거나 변경, 또는 제거할 수 있다.

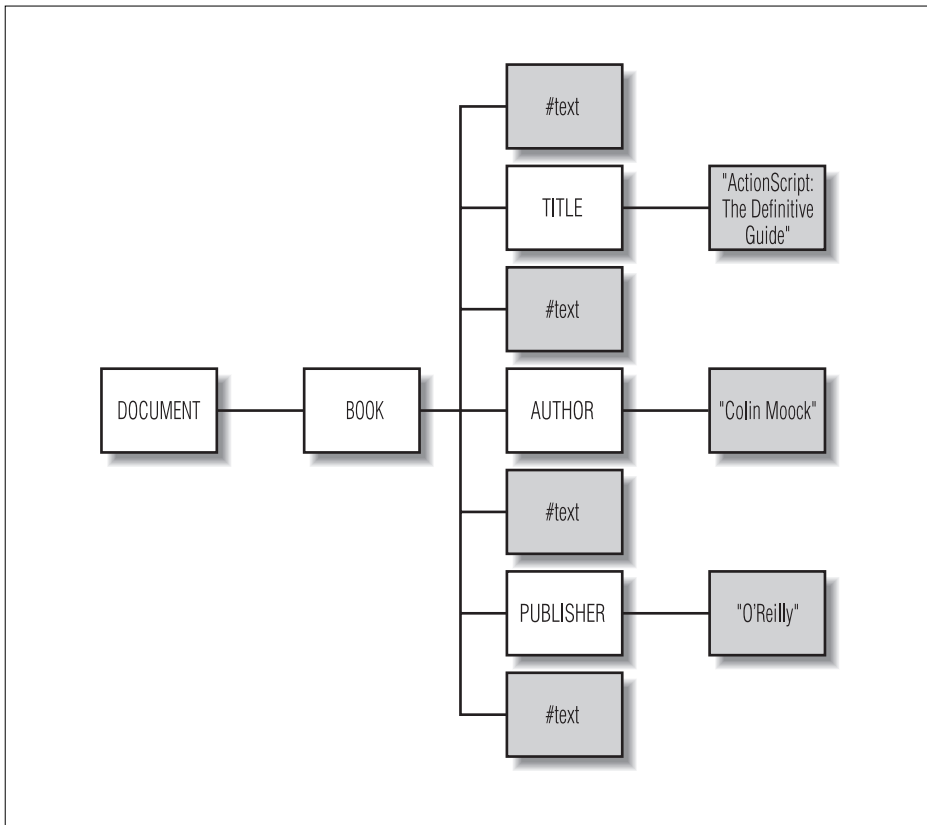
XML 문서의 소스 코드는 대부분 일련의 원소와 특성으로 이루어진다. 예를 들어 다음과 같은 XML 구문을 보면 BOOK, TITLE, AUTHOR, PUBLISHER는 우리에게 비교적 익숙한 HTML 태그와 똑같은 형식을 지니고 있으며, AUTHOR 원소에는 SALUTATION이라는 특성이 있다는 것을 알 수 있다.

```
<BOOK>
  <TITLE>ActionScript: The Definitive Guide</TITLE>
  <AUTHOR SALUTATION="Mr.">Colin Moock</AUTHOR>
  <PUBLISHER>O'Reilly</PUBLISHER>
</BOOK>
```

객체지향적인 관점에서 본다면 XML 문서의 내용을 각 원소와 텍스트 블록이 플로우차트와 비슷한 구조에서 객체 노드를 이루는 객체의 계층으로 다룰 수 있다. [그림 R-1]은 위 <BOOK> 구문을 XML 객체 계층 형식으로 나타낸 개념도이다.

이 샘플 XML 객체 계층의 구조와 의미를 왼쪽에서 오른쪽 방향으로 살펴보자. 우선 [그림 R-1]에서 DOCUMENT로 표시된 메인 XML 객체에서 시작해 보면, 이 객체는 XML 생성자에 의해 자동으로 생성되고 XML 객체 계층을 담아줄 컨테이너 역할을 한다. BOOK 노드는 이 XML 데이터 구조의 루트이다(모든 제대로 정의된 XML 문서에는 이 객체에서의 BOOK과 같이 다른 모든 원소를 포함하는 루트 원소가 있어야 한다). XML 객체 계층의 소스 코드를 파싱하여 만들 수도 있고 계층에 들어 있는 객체의 노드 추가 메소드를 호출하여 만들 수도 있다.

어떤 노드가 다른 노드에 포함되어 있으면 그 노드는 '자식(child)', 그 노드를 포함하고 있는 노드는 '부모(parent)'라고 부른다. 위 예제에서 BOOK은 DOCUMENT의 첫 번째 자식이고 DOCUMENT는 BOOK의 부모이다.



[그림 R-1] XML 노드 계층의 예

[그림 R-1]의 오른쪽 부분을 보면 BOOK에는 앞에 나온 XML 소스에는 보이지 않는 네 개의 #text 노드를 포함한 일곱 개의 자식이 있다는 것을 알 수 있다. XML 소스 코드에 있는 각 공백은 XML 객체 계층에서 객체 형태로 구현된다. 앞에 나온 XML 코드를 자세히 살펴보면 BOOK과 TITLE 사이에 공백(줄바꿈 문자 및 탭 문자)이 들어있다는 것을 알 수 있다. 이 공백과 TITLE, AUTHOR, PUBLISHER 사이에 각각 들어있는 공백은 [그림 R-1]에서 #text라는 노드로 표현된다.

BOOK의 자식들 사이에서는 ‘형제(sibling)’ 관계가 성립된다(즉 계층에서 같은 레벨에 들어간다). 예를 들어 AUTHOR 뒤에 있는 형제는 #text이고 앞에 있는 형제도 #text가 된다. 같은 계층에 속하는 형제들을 돌아다니면서 어떤 작업을 할 때 #text 노드가 방해 요인이 될 수도 있다. 이처럼 아무 내용도 없는 공백 노드는 다음과 같은 식으로 처리하면 된다.

- 일일이 찾아내서 객체 계층에서 제거한다(공백 제거 코드는 잠시 뒤에 나온다).
- 코드에서 공백 노드를 감지하고 건너뛰도록 만든다(노드 사이를 움직이는 방법은 nextSibling 및 previousSibling 부분 참조).
- 공백 노드가 아예 생기지 않도록 XML 소스 자체에서 공백을 제거한다.
- 파싱을 시작하기 전에 XML 객체의 ignoreWhite 속성을 true로 설정한다(플래시 5r41부터 사용 가능함).

계층의 가장 오른쪽 부분을 보면 TITLE, AUTHOR, PUBLISHER 노드에 각각 자식이 하나씩 있음을 알 수 있다. 각 자식은 TITLE, AUTHOR, PUBLISHER 원소에 포함되는 텍스트에 해당하는 텍스트 노드이다. XML 소스 코드의 원소에 포함된 텍스트는 그 원소의 자식 노드로 저장된다. 원소에 저장된 텍스트를 사용하고 싶다면 firstChild.nodeValue 또는 childNodes[0].nodeValue와 같은 식으로 그 원소의 자식을 참조해야 한다.

그렇다면 원소의 특성은 어떻게 다루어야 할까? XML 객체에서 원소의 특성이 포함되는 부분은 어디일까? AUTHOR의 SALUTATION 특성은 SALUTATION이라는 자식 노드로 저장될 것이라고 생각하는 독자도 있을 것이다. 하지만 어떤 원소에 포함되는 '특성(attribute)'은 그 원소의 자식이 아니라 그 원소의 '속성(property)'으로 간주된다. 특성 속성을 사용하는 방법은 XML.attributes 부분에서 살펴보기로 하자.

XML 문서를 노드 객체의 계층으로 만드는 방법을 알아보자. 아무 내용도 들어 있지 않은 XML 객체를 새로 만들 때는 XML() 생성자를 이용하면 된다.

```
myDocument = new XML();
```

그 다음에 appendChild(), parseXML(), load()와 같은 메소드를 호출하여 새로운 노드를 추가할 수 있다. 또는 XML 생성자를 호출할 때 source 인자를 사용하면 XML 소스로부터 XML 객체를 만들 수 있다.

```
myDocument = new XML(source);
```

예를 들면 다음과 같다.

```
myDocument = new XML('<P>hello world!</P>');
```

XML() 생성자를 호출할 때 source 인자를 전달하면 source를 파싱하여 새로운 객체 계층으로 변환하고 그 내용을 객체로 저장하여 생성자가 종료될 때 그 객체를 리턴한다(이 경우에는 P가 myDocument의 첫 번째 자식이 되며, nodeValue가 “hello world!”인 텍스트 노드가 P의 첫 번째 자식이 된다).

XML 계층을 만들고, 그 계층을 객체에 저장하고 나면 XML 클래스의 메소드와 속성을 이용하여 그 계층에 있는 정보를 액세스할 수 있다. 예를 들어 myDocument에서 “hello world!”라는 텍스트를 도출해야 하는 경우를 생각해 보자. 객체 지향적인 방법으로 생각해 본다면 P의 텍스트는 myDocument.P와 같이 myDocument의 속성으로 액세스할 수 있으면 될 것이다. 하지만 이렇게 하면 안 된다. 노드를 참조할 때는 이름 대신 firstChild나 childNodes와 같은 XML 내장 속성을 사용해야 한다. 예를 들어 다음과 같은 코드를 이용하여 P 노드를 액세스할 수 있다.

```
myDocument.firstChild      // P를 액세스하는 코드
myDocument.childNodes[0]   // P를 액세스하는 코드
```

firstChild는 주어진 노드의 첫 번째 자식에 대한 레퍼런스를 리턴하는 속성이므로 myDocument.firstChild는 P에 대한 레퍼런스를 리턴한다. 하지만 P 자체가 아니라 P에 들어있는 텍스트인 “hello world!”를 구하는 방법을 찾아야 한다. 따라서 다음과 같은 식으로 텍스트 노드(P의 첫 번째 자손)를 참조해야 한다.

```
myDocument.firstChild.firstChild // P 아래에 있는 텍스트 노드를
                                // 액세스한다.
```

노드의 값을 구하려면 nodeValue 속성을 이용하면 된다. 예를 들어 다음과 같이 하면 “hello world!”를 Output 창에 표시할 수 있다.

```
trace(myDocument.firstChild.firstChild.nodeValue);
```

P 아래에 있는 텍스트 노드의 값을 다음과 같이 바꿀 수도 있다.

```
myDocument.firstChild.firstChild.nodeValue = "goodbye cruel world";
```

P 노드를 통째로 바꾸거나 새로운 노드를 추가하거나 “hello world!”라는 텍스트를 다른 노드로 옮기고 싶다면, XML 클래스의 메소드를 이용하면 된다. 예를 들면 다음과 같다.

```
// P를 삭제한다.
myDocument.firstChild.removeNode();

// P라는 이름을 가진 새로운 원소를 만든다.
newElement = myDocument.createElement("P");

// 새로운 원소를 문서에 추가한다.
myDocument.appendChild(newElement);

// P에 추가할 새로운 텍스트 노드를 만든다.
newText = myDocument.createTextNode("XML is fun");

// 새로 만든 텍스트 노드를 P에 추가한다.
myDocument.firstChild.appendChild(newText);
```

위 예제에서 볼 수 있듯이 XML 구조를 가진 데이터에 대한 작업은 간접적인 방식으로 이루어진다. 데이터를 새로 만들거나 삭제하거나 조작하는 것은 객체의 메소드를 호출하거나 속성을 액세스하는 방법으로 처리된다. XML 데이터를 다루는데 필요한 다양한 도구에 대해 배우고 싶다면, 앞으로 나올 XML 클래스의 속성과 메소드에 대해 자세히 읽어보면 된다.

액션스크립트에서는 XML 데이터를 처리할 때 월드 와이드 웹 컨소시엄(W3C, World Wide Web Consortium)에서 발표한 문서 객체 모델(DOM, Document Object Model) 표준을 따른다. DOM에서 XML 구조를 가진 데이터를 객체 계층으로 표현하는 방법은 아래 URL에 자세히 나와 있다.

<http://www.w3.org/DOM>

언어와 상관없는 핵심 DOM 스펙은 아래 사이트에서 찾을 수 있다.

<http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html>

ECMA-262에서 DOM을 구현하는 것과 관련된 내용은 다음 사이트에 나와 있다.

<http://www.w3.org/TR/REC-DOM-Level-1/ecma-script-language-binding.html>

예제

XML 소스 코드에서 두 개의 원소 사이에 공백이 들어가면 그 공백 때문에 XML 객체 계층에 텍스트 노드가 추가된다. 플래시 5 플레이어 빌드 41 전에는 XML 객체 계층이 생성된 후에 수동으로 모든 공백 노드를 제거해야 한다. XML을 다루다 보면 특별한 종류의 노드를 제거하는 일을 자주 하게 되며, 이러한 작업은 트리 횡단(tree traversal, 계층에 있는 모든 노드를 거쳐가는 작업)의 좋은 예가 된다. 문서에 있는 공백 노드를 제거하는 두 가지 방법에 대해 알아보자.

첫 번째 예제에서는 FIFO(First In First Out, 선입선출) 스택을 이용하여 트리에 있는 모든 노드를 배열에 집어넣어 처리한다. `stripWhitespaceTraverse()` 함수에서는 우선 인자로 받아들인 `theNode`를 원소로 하는 배열을 만든다. 그리고 나서 배열의 첫 번째 노드를 제거하고 그 자식을 배열에 추가한다. 배열에 더 이상 원소가 없으면 `theNode`의 모든 자손 노드가 처리된 것이다. 처리과정에서 자식이 하나도 없는 노드는 공백 노드일 가능성이 높은 것으로 간주한다(텍스트 노드에는 자식이 없기 때문이다). 이러한 노드를 모두 살펴보고 다음과 같은 조건에 해당하는지 확인한다.

- 텍스트 노드(`nodeType` 속성으로 확인)
- 공백으로 간주되지 않는 ASCII 32 이상의 문자를 포함하는 노드

이러한 과정을 거쳐 ASCII 32 미만의 문자(즉 공백문자)로만 이루어진 텍스트 노드를 삭제한다.

```
// FIFO 스택을 이용하여 공백을 제거하는 코드
// 트리를 횡단하여 theNode 아래에 있는 모든 공백 노드를 제거한다.
function stripWhitespaceTraverse (theNode) {
    // 처리할 노드의 목록을 만든다.
    var nodeList = new Array();
    // 함수에 전달된 노드로부터 배열을 채운다.
    nodeList[0] = theNode;

    // 처리할 자손 노드가 없을 때까지 노드를 처리한다.
    while (nodeList.length > 0) {
        // 목록에서 첫 번째 노드를 제거하고 그 노드를 currentNode에 대입한다.
        currentNode = nodeList.shift();

        // 노드에 자식이 있는 경우
        if (currentNode.childNodes.length > 0) {
            // 이 노드의 자식 노드들을 처리할 노드의 목록에 추가한다.
```

```

        nodeList = nodeList.concat(currentNode.childNodes);
    } else {
        // 그렇지 않다면 그 노드가 가지의 끝에 있는 노드이므로
        // 텍스트 노드인지 확인한다. 만약 그렇다면 공백문자만
        // 들어있는지 확인한다. nodeType 3은 텍스트 노드를 의미한다.
        if (currentNode.nodeType == 3) {
            var i = 0;
            var emptyNode = true;
            for (i = 0; i < currentNode.nodeValue.length; i++) {
                // ASCII 코드가 32 미만이면 필요 없는 공백
                // 문자로 간주한다.
                if (currentNode.nodeValue.charCodeAt(i) > 32) {
                    emptyNode = false;
                    break;
                }
            }

            // 모두 공백문자이면 그 노드를 제거한다.
            if (emptyNode) {
                currentNode.removeNode();
            }
        }
    }
}
}
}

```

위 예제에서 사용한 방법은 전통적으로 그 효율성이 인정된 방법이다. 하지만 플래시 5 플레이어에서는 `Array.concat()` 메소드의 실행 속도가 매우 느리다. 따라서 다음 예제에 나온 기법을 이용하여 공백을 제거하는 것이 속도 면에서 더 유리하다. 주석을 자세히 읽어보자.

```

// 함수 재귀 호출을 이용하여 공백을 제거한다.
// 트리의 각 레벨을 두 번씩 지나가면서 XML 문서의
// 공백 노드를 제거한다.
function stripWhitespaceDoublePass(theNode) {
    // theNode의 모든 자식 노드에 대해 루프를 돌린다.
    for (var i = 0; i < theNode.childNodes.length; i++) {
        // 현재 노드가 텍스트 노드이면
        if (theNode.childNodes[i].nodeType == 3) {
            // 노드에 공백문자가 아닌 문자가 있는지 확인한다.
            var j = 0;

```

```
var emptyNode = true;
for (j = 0; j < theNode.childNodes[i].nodeValue.length; j++) {
    // ASCII 32 미만의 문자는 모두 공백문자에 해당한다.
    if (theNode.childNodes[i].nodeValue.charCodeAt(j) > 32) {
        emptyNode = false;
        break;
    }
}

// 모두 공백문자라면 그 노드를 제거한다.
if (emptyNode) {
    theNode.childNodes[i].removeNode();
}
}

// theNode에 있는 공백 노드는 모두 제거했으므로 나머지 자식에 대해
// stripWhitespaceDoublePass()를 재귀적으로 호출한다.
for (var k = 0; k < theNode.childNodes.length; k++) {
    stripWhitespaceDoublePass(theNode.childNodes[k]);
}
}
```

참조

XMLnode 클래스, XMLSocket 클래스, 18장의 'HTML 지원'

XML.appendChild() 메소드

노드에 새로운 자식 노드를 추가하거나
원래 있던 노드를 이동시키는 메소드

버전 플래시 5

문법 *theNode.appendChild(childNode)*

인자

childNode 현존하는 XML 노드 객체

설명

`appendChild()`는 주어진 `childNodes`를 `theNode`의 마지막 자식으로 추가하는 메소드이다. `appendChild()`는 원래 있는 노드에 새로운 노드를 추가하거나 같은 문서 내에서, 혹은 서로 다른 문서 사이에서 노드를 이동시키기 위한 용도로 쓰인다. `childNodes`에는 원래 있던 노드 객체에 대한 레퍼런스를 전달해야 한다.

원래 있던 노드에 새로운 자식 노드를 추가하려면, 우선 `createElement()`, `createTextNode()` 또는 `cloneNode()`를 사용하거나 XML 소스 코드를 XML 객체로 파싱하여 새로운 노드를 만들어야 한다. 예를 들어 다음과 같은 코드를 이용하면, 새로운 P 노드와 텍스트 노드를 만들고 텍스트 노드를 P 노드에 붙인 다음에 P 노드를 문서의 최상위 노드에 P 노드를 붙인다.

```
// 문서를 만든다.
myDoc = new XML('<P>paragraph 1</P>');

// P 노드와 텍스트 노드를 만든다.
newP = myDoc.createElement("P");
newText = myDoc.createTextNode("paragraph 2");

// 텍스트 노드를 P 노드에 붙인다.
newP.appendChild(newText);

// P 노드를 myDoc에 추가한다(자식도 함께 추가됨).
myDoc.appendChild(newP);

trace(myDoc); // "<P>paragraph 1</P><P>paragraph 2</P>"라고 출력된다.
```

문서 내에서 노드를 이동시킬 때는 그 문서 안에 있는 노드에 대한 레퍼런스를 `childNodes`로 지정하면 된다. 이 때 `childNodes`는 노드의 처음 위치를, `theNode`는 그 노드의 새로운 부모 노드를 의미한다. 이 과정에서 `childNodes`는 원래의 부모 노드에서 제거된다. 예를 들어 다음 코드에서는 B 노드를 원래 부모 노드였던 P 노드에서 제거하고 문서의 루트로 이동시킨다.

```
// 새로운 문서를 만든다.
myDoc = new XML('<P>paragraph 1<B>bold text</B></P>');

// B 노드에 대한 레퍼런스를 저장한다.
boldText = myDoc.firstChild.childNodes[1];
```



```
// B 노드를 P 노드에서 삭제하고 문서의 루트로 옮긴다.
```

```
myDoc.appendChild(boldText);
```

```
trace(myDoc); // "<P>paragraph 1</P><B>bold text</B>"라고 출력된다.
```

레퍼런스를 저장하는 단계를 건너뛰고 다음과 같은 식으로 해도 된다.

```
myDoc.appendChild(myDoc.firstChild.childNodes[1]);
```

서로 다른 문서 사이에서 노드를 이동할 때는 `childNodes`에는 첫 번째(원본) 문서에 있는 노드에 대한 레퍼런스, `theNode`에는 두 번째(대상) 문서에 있는 노드에 대한 레퍼런스를 사용해야 한다. 예를 들어 다음과 같이 하면 `myDoc`에 있는 B 노드를 `myOtherDoc`으로 옮길 수 있다.

```
myDoc = new XML('<P>paragraph 1<B>bold text</B></P>');
```

```
myOtherDoc = new XML();
```

```
myOtherDoc.appendChild(myDoc.firstChild.childNodes[1]);
```

```
trace(myDoc); // "<P>paragraph 1</P>"라고 출력된다.
```

```
trace(myOtherDoc); // "<B>bold text</B>"라고 출력된다.
```

참조

`XML.createElement()`, `XML.createTextNode()`, `XML.cloneNode()`,
`XML.insertBefore()`

XML.attributes 속성

원소 특성을 속성으로 저장하는 객체

버전 플래시 5

문법 `theNode.attributes.attributeIdentifier`
 `theNode.attributes[attributeNameInQuotes]`

액세스 읽기/쓰기

설명

attributes 속성에는 theNode에서 정의한 특성의 이름과 값이 속성 형태로 저장된다. 예를 들어 다음과 같은 P 태그의 ALIGN 특성은

```
<P ALIGN="CENTER">this is a paragraph</P>
```

theNode.attributes.ALIGN 또는 theNode.attributes["ALIGN"]과 같은 식으로 액세스하면 된다. XML 소스에 P 태그만 있다면 다음과 같은 식으로 ALIGN 특성을 액세스할 수도 있다.

```
// XML 객체 계층을 만든다.
myDoc = new XML('<P ALIGN="CENTER">this is a paragraph</P>');

// ALIGN 특성을 액세스한다. "CENTER"가 출력된다.
trace(myDoc.firstChild.attributes.ALIGN);

// ALIGN 특성을 설정한다.
myDoc.firstChild.attributes.ALIGN = "LEFT";
```

attributes 속성은 객체이다. 다음과 같이 theNode에 새로운 특성을 추가하면 attributes에 새로운 속성이 추가된다.

```
// P 태그에 CLASS 특성을 추가한다.
myDoc.firstChild.attributes.CLASS = "INTRO";

// firstChild는 XML 소스를 가리킨다.
// <P ALIGN="CENTER" CLASS="INTRO">this is a paragraph</P>
```

attributes는 배열이 아니기 때문에 length 속성은 사용할 수 없다. 대신 for-in 루프를 이용하면 원소에 있는 모든 특성을 액세스할 수 있다.

```
var count = 0;
for(var prop in theNode.attributes) {
    trace("attribute" + prop + "has the value" + theNode.
        attributes[prop]);
    count++;
}
trace ("The node has" + count + "attributes.");
```

theNode가 가리키는 XML 원소에는 특성이 하나도 없기 때문에, attributes 속성은 아무런 내용도 없는 비어있는 객체가 되고 따라서 위 예제를 실행하면 0개의 특성이 있다고 나올 것이다.

참조

XML.nodeType

XML.childNodes 속성

노드의 자식에 대한 레퍼런스의 배열

버전	플래시 5
문법	<i>theNode</i> .childNodes[n]
액세스	읽기 전용

설명

childNodes는 theNode의 모든 직계 자손에 대한 레퍼런스가 원소로 저장된 배열 형태의 속성이다. 이 속성은 XML 계층을 따라 노드를 액세스하기 위한 용도로 쓰인다. 예를 들어 다음과 같은 식으로 객체 계층을 만들면

```
myDoc = new XML('<STUDENT><NAME>Tim</NAME><MAJOR>BIOLOGY</MAJOR></STUDENT>');
```

다음과 같은 식으로 STUDENT 노드를 액세스할 수 있다.

```
myDoc.childNodes[0];
```

NAME과 MAJOR 노드(STUDNET의 자식)는 다음과 같은 식으로 액세스하면 된다.

```
myDoc.childNodes[0].childNodes[0];    // NAME
myDoc.childNodes[0].childNodes[1];    // MAJOR
```

theNode 아래의 계층이 변경되면 childNodes도 자동으로 업데이트되어 새로운 구조가 그대로 반영된다. 예를 들어 MAJOR 노드를 삭제하면 myDoc.childNodes[0].childNodes[1]에서 undefined를 리턴한다.

정보를 처리하거나 문서의 구조를 변경하기 위한 용도로 노드를 참조할 수도 있다. 예를 들어 다음과 같은 식으로 학생의 이름을 바꾸거나 새로운 학생을 추가할 수 있다.

```
// 학생의 이름을 확인한다.
trace("The student's name is: "
      + myDoc.childNodes[0].childNodes[0].childNodes[0].nodeValue);

// 학생의 이름을 바꾼다.
myDoc.childNodes[0].childNodes[0].childNodes[0].nodeValue = "James";

// STUDENT 노드를 복사한다.
newStudent = myDoc.childNodes[0].cloneNode(true);

// 문서에 새로운 STUDENT 노드를 추가한다.
myDoc.appendChild(newStudent);
```

편의상 `childNodes[0]` 대신 `firstChild` 속성을 사용해도 된다. 따라서 아래 나오는 두 가지 레퍼런스는 똑같은 노드를 가리키게 된다.

```
myDoc.childNodes[0];
myDoc.firstChild;
```

노드의 모든 자식에 대해 순환문을 돌릴 때는 다음과 같은 식으로 `for` 문을 사용하면 된다.

```
for (var i = 0; i < theNode.childNodes.length; i++) {
    trace("child" + i + "is" + theNode.childNodes[i].nodeName);
}
```

하지만 이 방법으로는 `theNode`의 계층에서 첫 번째 레벨밖에 횡단하지 못한다. `theNode` 이하의 모든 노드를 액세스할 수 있는 방법은 `XML.nextSibling` 항목에 나온 예제에 수록하였다. `theNode`에 자식이 하나도 없으면 `theNode.childNodes.length` 속성이 0이 된다.

주의 사항

XML 소스 코드에서 사용한 공백 때문에 생기는 비어있는 텍스트 노드도 `childNodes` 목록에 포함된다는 점에 주의하자. 예를 들어 다음과 같은 XML 소스에

서는 BOOK을 시작하는 태그와 TITLE, AUTHOR, PUBLISHER를 끝마치는 태그 뒤에 있는 공백 때문에 비어있는 텍스트 노드가 생긴다.

```
<BOOK>
  <TITLE>ActionScript: The Definitive Guide</TITLE>
  <AUTHOR SALUTATION="Mr">Colin Moock</AUTHOR>
  <PUBLISHER>O'reilly</PUBLISHER>
</BOOK>
```

따라서 BOOK의 첫 번째 자식 노드는 비어있는 텍스트 노드이고 두 번째 자식 노드가 TITLE이 된다.

참조

XML.firstChild, XML.hasChildNodes(), XML.lastChild, XML.nextSibling, XML.previousSibling

XML.cloneNode() 메소드

노드를 복사하는 메소드

버전 플래시 5

문법 *theNode.cloneNode(deep)*

인자

deep theNode의 자식을 재귀적으로 복사할지 나타내는 부울 값. true이면 theNode 이하의 모든 계층을 복사하고 false이면 theNode 자체만(원소 노드이면 그 특성도 포함) 복사한다.

리턴 값

theNode 객체의 복사본. deep 인자가 true이면 서브트리도 함께 복사된다.

설명

cloneNode() 메소드에서는 theNode의 복사본(theNode가 원소 노드인 경우에는 특성과 값도 포함)을 만들어서 리턴한다. deep이 true이면 theNode 이하의 모든 노드 계층들이 함께 복사된다.

cloneNode()는 보통 원래 있던 템플릿을 기반으로 새로운 노드를 만들기 위한 용도로 쓰인다(이렇게 하면 새로운 노드 구조를 수동으로 일일이 만들지 않아도 된다). 노드를 복사하고 나면 보통 그 노드를 용도에 맞게 적절히 수정한 다음, appendChild()나 insertBefore() 메소드를 이용하여 XML 문서에 집어넣는다. 아래 예제에서는 문서의 첫 번째 문단을 복사하여 똑같은 구조를 가지는 형제 노드를 만든다.

```
// 새로운 문서를 만든다.
myDoc = new XML('<P>paragraph 1</P>');

// 첫 번째 문단을 복사한다.
newP = myDoc.firstChild.cloneNode(true);

// 노드를 수정한다.
newP.firstChild.nodeValue = "paragraph 2";

// 복사본을 원래 문서에 집어넣는다.
myDoc.appendChild(newP);

trace(myDoc); // "<P>paragraph 1</P><P>paragraph 2</P>"가 출력된다.
```

원소에 있는 텍스트는 그 원소와는 별도로 자식 노드에 저장되므로 원소의 텍스트 내용도 함께 복사할 때는 deep 인자를 true로 설정해야 한다는 점에 주의하자. cloneNode()에서 리턴하는 원소는 자동으로 원래 문서에 추가되지 않기 때문에, appendChild() 또는 insertBefore()를 이용하여 직접 문서에 추가하는 작업을 처리해야 한다.

참조

XML.appendChild(), XML.createElement(), XML.createTextNode(), XML.insertBefore()

XML.contentType 속성

XML.send()나 XML.sendAndLoad()를 이용하여 보내는 XML 데이터의 MIME 유형

버전 플래시 5 빌드 41 이후

문법 XMLdoc.contentType

액세스 읽기/쓰기

설명

contentType 속성은 XML.send()나 XML.sendAndLoad()를 호출했을 때 서버로 전송되는 MIME 유형을 나타낸다. 이 속성의 기본값은 application/x-www-urlform-encoded이다. 특정 XML 객체의 contentType 속성을 수정하여 그 객체의 MIME 유형만 수정할 수도 있지만, XML.prototype.contentType을 고치면 모든 XML 객체의 MIME 유형을 바꿀 수 있다.

contentType 속성은 플래시 5 플레이어의 빌드 41부터 도입되었기 때문에, 그 이전 버전에서는 MIME 유형을 따로 설정할 수가 없다. contentType을 undefined와 비교하거나 getVersion()을 이용하여 플레이어 버전과 빌드 번호를 확인하여 이 속성을 사용할 수 있는지 미리 확인해 보는 것이 좋다.

주의 사항

XML.send() 항목에서 MIME 유형을 설정하는 것과 관련된 내용 참조

참조

XML.send(), XML.sendAndLoad()

XML.createElement() 메소드

새로운 원소 노드를 만드는 메소드

버전 플래시 5

문법 *XMLdoc.createElement (tagName)*

인자

tagName 만들 원소의 이름을 나타내는 문자열(대소문자를 구분함). 예를 들어 <P ALIGN="RIGHT">라는 태그에서는 P가 태그 이름이 된다.

리턴 값

부모나 자식이 없는 새로운 원소 노드 객체

설명

createElement()는 XML 문서 객체 계층에 추가할 새로운 원소 노드를 만들 때 가장 많이 사용하는 메소드이다. createElemnt() 메소드에서는 원소를 리턴할 때 자동으로 XMLdoc 객체에 그 원소를 추가하지 않기 때문에, appendChild()나 insertBefore()를 이용하여 직접 XML 계층에 넣어주어야 한다는 점에 주의하자. 아래 예에서는 문서에 새로운 P 원소를 추가한다.

```
myDoc = new XML();
newP = myDoc.createElement("P");
myDoc.appendChild(newP);
```

위 코드에서 세 단계에 걸쳐 한 작업을 다음과 같이 코드 한 줄로 처리할 수도 있다.

```
myDoc.appendChild(myDoc.createElement("P"));
```

이 때 XMLdoc은 XMLnode가 아니라 XML 클래스에 속하는 인스턴스여야 한다는 점에 주의하자.

createElement() 메소드는 텍스트 노드를 만들 때는 사용할 수 없다. 텍스트 노드는 createTextNode() 메소드로 만들어야 한다.

참조

XML.appendChild(), XML.cloneNode(), XML.createTextNode(), XML.insertBefore()

XML.createTextNode() 메소드

새로운 텍스트 노드를 만드는 메소드

버전 플래시 5

문법 *XMLdoc*.createTextNode(*text*)

인자

text 새로운 노드의 nodeValue로 들어갈 텍스트를 포함하는 문자열

리턴 값

부모나 자식이 없는 새로운 텍스트 노드 객체

설명

`createTextNode()`는 XML 문서 계층에 추가할 새로운 텍스트 노드를 만들 때 가장 많이 쓰이는 메소드이다. `createTextNode()` 메소드에서는 객체를 리턴할 때 자동으로 `XMLDoc`에 그 객체를 추가하지 않기 때문에, `appendChild()`나 `insertBefore()`를 이용하여 직접 XML 계층에 집어넣어야 한다는 점에 주의하자. 예를 들어 아래 예제에서는 새로운 P 원소를 문서에 추가하고 나서 그 P 원소에 텍스트 노드 자식을 집어넣는다.

```
myDoc = new XML();
newP = myDoc.createElement("P");
myDoc.appendChild(newP);

newText = myDoc.createTextNode("This is the first paragraph");
myDoc.firstChild.appendChild(newText);

trace(myDoc); // "<P>This is the first paragraph</P>"가 출력된다.
```

이 때 `XMLDoc`은 `XMLNode`가 아니라 XML 클래스에 속하는 인스턴스여야 한다는 점에 주의하자.

텍스트 노드는 보통 `createElement()`를 이용하여 만드는 원소 노드의 자식으로 저장된다.

참조

`XML.appendChild()`, `XML.cloneNode()`, `XML.createElement()`, `XML.insertBefore()`

XML.docTypeDecl 속성

문서의 DOCTYPE 태그

버전	플래시 5
문법	<code>XMLDoc.docTypeDecl</code>
액세스	읽기/쓰기

설명

docTypeDecl은 XMLdoc의 DOCTYPE 태그가 존재한다면 그 태그를 나타내는 문자열 속성이다. 만약 그 태그가 없다면 undefined 값을 가지게 된다. XMLdoc은 XML 객체 계층(즉 XMLnode가 아닌 XML 클래스의 인스턴스)에서 최상위 노드에 해당한다.

XML 문서의 DOCTYPE은 그 문서를 검증하기 위해 필요한 DTD의 이름과 위치를 나타낸다. 액션스크립트에서는 XML 문서를 검증하지는 않고 파싱만 한다. DOCTYPE 태그는 외부에서 검증할 수 있는 XML 문서를 만들거나 XML 문서의 유형을 알아내기 위한 용도로 쓰인다.

예제

```
var myXML = new XML('<?XML version="1.0"?>
    <!DOCTYPE foo SYSTEM "bar.dtd">' + '<P>a short document</P>');

trace(myXML.docTypeDecl); // "<!DOCTYPE foo SYSTEM "bar.dtd">"라고
                          // 출력된다.

// DOCTYPE을 새로 설정한다.
myXML.docTypeDecl = '<!DOCTYPE baz SYSTEM "bam.dtd">';
```

참조

XML.XMLDecl

XML.firstChild 속성

노드의 첫 번째 자손에 대한 레퍼런스

버전	플래시 5
문법	<i>theNode</i> .firstChild
액세스	읽기 전용

설명

firstChild 속성은 childNodes[0]와 같다. 따라서 theNode 아래에 있는 첫 번째 노드 객체에 대한 레퍼런스를 리턴한다. theNodedp 자식이 없다면 firstChild에서 null을 리턴한다.

다음 XML 소스에서 MESSAGE 노드의 firstChild는 nodeValue가 “hey”인 텍스트 노드가 된다.

```
<!-- Fragment 1 -->
<MESSAGE>hey</MESSAGE>
```

또한 다음과 같은 XML 소스에서는 ROOM 노드가 HOTEL 노드의 firstChild가 된다.

```
<!-- Fragment 2 -->
<HOTEL><ROOM><SIZE>Double</SIZE></ROOM></HOTEL>
```

theNode가 객체 계층에서 제일 위에 있을 때(즉 XML 문서 객체를 가리킬 때) firstChild, 즉 문서에 있는 첫 번째 원소가 유용한 원소가 아닐 수도 있다. 문서에 XML 정의 부분(<?XML version="1.0"?>)과 DOCTYPE 태그가 들어갈 가능성이 높기 때문에, XML 계층의 루트 원소 앞에 공백 원소가 들어가는 경우가 많다. 하지만 XML 구문에 XML 정의 부분과 DOCTYPE 태그가 없다면 firstChild를 문서의 첫 번째 노드로 간주할 수 있다.

```
// 새로운 XML 객체를 만든다.
myDoc = new XML('<MESSAGE><USER>gray</USER><CONTENT>hi</CONTENT>
</MESSAGE>');

// XML 객체의 첫 번째 노드를 msg라는 변수에 저장한다.
msg = myDoc.firstChild;

// USER 태그에 포함된 텍스트를 userNameOutput
// 이라는 텍스트 필드에 대입한다.
userNameOutput = msg.firstChild.firstChild.nodeValue;
```

USER 태그에 포함된 텍스트를 액세스하기 위해 nodeValue를 사용하는 방식은 형식을 제대로 따르는 것이긴 하지만, 꼭 그렇게 할 필요는 없다. 문자열이 들어갈 자리에 텍스트 노드 객체를 사용하면 그 노드에 대해 toString() 메소드를 자동으로 호출하여 노드에 있는 텍스트가 리턴되기 때문이다.

참조

XML.childNodes, XML.lastChild, XML.nextSibling, XML.previousSibling

XML.hasChildNodes() 메소드

노드에 자손이 있는지 확인하기 위한 메소드

버전 플래시 5**문법** `theNode.hasChildNodes()`**리턴 값**

부울 값. theNode의 자식이 있으면 true를, 없으면 false를 리턴한다.

설명

hasChildNodes() 메소드는 주어진 노드에서 다른 노드 계층이 나오는지 확인하기 위한 메소드이다. 다음과 같은 비교 구문을 사용하는 것과 같은 결과를 리턴한다.

```
theNode.childNodes.length > 0
```

theNode에 서브노드가 없으면 false를 리턴한다.

예제

노드를 처리할 때 hasChildNodes()를 이용하여 그 노드를 처리할지 결정해야 하는 경우도 있다. 예를 들어 아래 코드에서는 첫 번째 자식에 더 이상 자식이 없을 때까지 첫 번째 자식 이하의 노드를 모두 제거한다.

```
while (myDoc.firstChild.hasChildNodes()) {
    myDoc.firstChild.firstChild.removeNode();
}
```

참조

XML.childNodes

XML.ignoreWhite 속성

XML을 파싱할 때 공백을 무시할지 결정하기 위한 속성

버전 플래시 5 빌드 41 이후**문법** `XMLdoc.ignoreWhite`**액세스** 읽기/쓰기

설명

`ignoreWhite` 속성에는 파싱 과정에서 공백만을 포함하는 텍스트 노드를 무시할지 결정하기 위한 부울 값이 저장된다. 기본값은 `false`(즉 공백 노드를 제거하지 않음)이다. 이 속성에서 설정한 내용은 특정 노드뿐만 아니라 XML 문서 전체에 적용된다. `XMLNode` 클래스의 인스턴스에서는 `ignoreWhite` 속성을 지원하지 않는다.

예제

특정 XML 문서에서만 파싱할 때 공백을 무시하도록 하려면 다음과 같은 구문을 사용한다.

```
myXML.ignoreWhite = true;
```

다음과 같이 하면 모든 XML 문서에서 공백 노드를 무시하도록 설정할 수 있다.

```
XML.prototype.ignoreWhite = true;
```

`ignoreWhite` 속성은 반드시 XML 문서를 파싱하기 전에 설정해야 한다(보통 `load()`나 `sendAndLoad()` 메소드를 호출하기 전에 설정한다).

참조

수동으로 공백을 제거하는 코드는 XML 클래스 항목의 예제로 나와 있다.

XML.insertBefore() 메소드

노드 앞에 새로운 형제 노드를 추가하는 메소드

버전 플래시 5

문법 `theNode.insertBefore(newChild, beforeChild)`

인자

newChild XML 노드 객체

beforeChild `beforeChild` 노드(`theNode`의 자식 노드) 앞에 `newChild`를 삽입한다.

설명

`insertBefore()` 메소드에서는 `newChild`를 `theNode`의 자식 목록에 추가하는데, 이 때 `beforeChild` 앞에 `newChild`를 집어넣는다. `insertBefore()` 메소드는

appendChild() 메소드와 유사하지만 XML 객체 계층에서 새로운 노드를 추가할 위치를 정확하게 지정할 수 있다는 점에서 차이가 난다.

예제

```
// 문서를 만든다.
myDoc = new XML('<P>paragraph 2</P>');

// P 노드와 텍스트 노드를 새로 만든다.
newP = myDoc.createElement("P");
newText = myDoc.createTextNode("paragraph 1");

// 텍스트 노드를 P 노드에 추가한다.
newP.appendChild(newText);

// 새로운 P 노드(텍스트 노드도 포함)를 원래 있던 P 노드 앞에 추가한다.
myDoc.insertBefore(newP, myDoc.firstChild);

trace(myDoc); // "<P>paragraph 1</P><P>paragraph 2</P>"가 출력된다.
```

참조

XML.appendChild()

XML.lastChild 속성

노드의 마지막 자손에 대한 레퍼런스

버전	플래시 5
문법	<i>theNode</i> .lastChild
액세스	읽기 전용

설명

lastChild 속성은 childNodes[childNodes.length-1]과 같다. 즉 theNode에서 나오는 마지막 노드 객체에 대한 레퍼런스를 리턴한다. theNode에 자식이 없으면 null을 리턴한다.

다음과 같은 XML 소스를 사용하면 MESSAGE 노드의 lastChild는 CONTENT 노드가 된다.

```
<MESSAGE><USER>gray</USER><CONTENT>hi</CONTENT></MESSAGE>
```

예제

```
// 새로운 XML 문서를 만든다.
myDoc = new XML('<MESSAGE><USER>gray</USER><CONTENT>hi</CONTENT>
               </MESSAGE>');

// myDoc의 firstChild는 MESSAGE이고 MESSAGE의
// lastChild는 CONTENT이고 CONTENT의 firstChild는
// "hi"라는 값을 가지는 텍스트 노드이므로 msg는 "hi"가 된다.
msg = myDoc.firstChild.lastChild.firstChild.nodeValue
```

참조

XML.childNodes, XML.firstChild, XML.nextSibling, XML.previousSibling

XML.load() 메소드

외부 문서에서 XML 소스 코드를 불러오는 메소드

버전 플래시 5

문법 `XMLdoc.load(URL)`

인자

URL 불러올 XML 문서의 위치를 나타내는 문자열

설명

load()는 외부 XML 문서를 불러오고 파싱하고 XML 객체 계층으로 변환하여 그 계층을 XMLdoc에 저장하는 메소드이다. 이전에 XMLdoc에 들어있던 내용은 새로 불러온 XML 문서로 대체된다.

XMLdoc은 XML 클래스가 아닌 XMLdoc 클래스에 속하는 인스턴스라는 점에 주의하자.

주의 사항

load() 메소드로 불러온 내용을 액세스하기 전에 로딩 및 파싱 작업이 제대로 처리되었는지 확인해야 한다. 확인할 때는 XML 문서의 loaded 속성 값을 확인해 보거나 로딩이 완료된 것을 감지할 수 있도록 문서에 onLoad() 콜백 핸들러를 대입하면 된다. 자세한 내용은 XML.loaded와 XML.onLoad() 항목에서 찾을 수 있다. 로딩된 데이터가 제대로 파싱되었는지 확인할 때는 문서의 status 속성을 이용한다.

XML.load()는 loadVariables() 전역 함수에 나온 [표 R-8]에 나온 도메인 기반의 보안 관련 제약 조건을 따른다.

예제

```
myDoc = new XML();
myDoc.load("myData.XML");
```

참조

XML.loaded, XML.onLoad(), XML.sendAndLoad(), XML.status

XML.loaded 속성

load()와 sendAndLoad() 메소드의 처리 상황

버전 플래시 5

문법 XMLdoc.loaded

액세스 읽기 전용

설명

loaded 속성에서는 전에 호출한 load()나 sendAndLoad() 작업이 완료되었는지를 나타내는 부울 값을 리턴한다. XML의 load()나 sendAndLoad() 메소드를 호출하면 이 값이 바로 false가 된다. 그리고 나서 로딩이 완료되면 loaded 값이 true로 바뀐다. 만약 XMLdoc에 대해 load()나 sendAndLoad() 메소드를 호출한 적이 없다면 loaded 값은 undefined이다.

loaded가 false이면 XML 데이터를 다운로드하고, 파싱하는 작업이 계속 진행 중임을 나타내면 XMLdoc의 객체 계층을 액세스할 수 없다. loaded가 true이면 XML 데이터의 다운로드 및 파싱 과정이 완료되어 데이터가 XMLdoc에 객체 계층 형태로 저장되어 있음을 나타낸다. 하지만 로딩한 XML 데이터가 제대로 파싱되어 있지 않을 수도 있다는 점에 주의하자(XMLdoc.status 속성을 이용하여 제대로 파싱되었는지 확인할 수 있다).

XMLdoc은 XML 클래스가 아닌 XMLnode 클래스의 인스턴스라는 점에 주의하자.

예제

아래 코드는 XML 데이터를 화면에 표시하기 전에 XML 데이터를 모두 불러올 때까지 기다리는 데 사용되는 간단한 XML 프리로더의 예이다(XML.onLoad() 핸들러를 이용하여 XML 프리로더를 만들 수도 있다).

```
// 1번 프레임에 들어가는 코드
```

```
// 새로운 XML 문서를 만든다.
```

```
myDoc = new XML();
```

```
// 외부 XML 파일을 문서로 불러온다.
```

```
myDoc.load("userProfile.XML");
```

```
// 5번 프레임의 코드
```

```
// 데이터 로딩이 완료되었는지 확인한다. 완료되고 나면
```

```
// 프레임을 화면에 표시하고 그렇지 않으면 4번 프레임으로
```

```
// 돌아간다. 데이터 로딩이 완료될 때까지 계속 반복한다.
```

```
if (myDoc.loaded) {
```

```
    if (myDoc.status == 0) {
```

```
        gotoAndStop("displayData");
```

```
    } else {
```

```
        gotoAndStop("loadingError");
```

```
    }
```

```
} else {
```

```
    gotoAndPlay(4);
```

```
}
```

참조

XML.load(), XML.onLoad(), XML.sendAndLoad()

XML.nextSibling 속성

바로 다음 노드에 대한 레퍼런스

버전 플래시 5

문법 *theNode.nextSibling*

액세스 읽기 전용

설명

nextSibling은 XML 객체 계층의 현 레벨에서 theNode 바로 뒤에 오는 노드 객체에 대한 레퍼런스를 리턴하는 속성이다. theNode 뒤에 노드가 하나도 없으면

nextSibling에서는 null을 리턴한다. 아래 XML 소스에서 CONTENT 노드는 USER 노드의 nextSibling이 된다.

```
<MESSAGE><USER>gray</USER><CONTENT>hi</CONTENT></MESSAGE>
```

예제

nextSibling 속성은 보통 XML 객체 계층을 횡단하는 데 쓰인다. 예를 들어 theNode의 모든 자식을 순서대로 출력하고 싶다면 다음과 같은 코드를 사용하면 된다.

```
for (var child = theNode.firstChild; child != null; child =
child.nextSibling) {
    trace("found node:" + child.nodeName);
}
```

다음과 같이 루프를 함수로 확장하면 XML 객체 계층에 있는 모든 노드를 재귀적으로 알아낼 수 있다.

```
function showNodes (node) {
    trace(node.nodeName + ": " + node.nodeValue);
    for (var child = node.firstChild; child != null; child =
child.nextSibling) {
        showNodes(child);
    }
}
```

```
// 노드 또는 문서에 대해 함수를 호출한다.
showNodes(myDoc);
```

위 두 예제에서 텍스트 노드는 nodeName 속성 항목에 설명한 것처럼 이름이 없는 형태로 나타난다.

참조

XML.childNodes, XML.firstChild, XML.lastChild, XML.nodeName, XML.nodeValue, XML.previousSibling

XML.nodeName 속성

현재 노드의 이름

버전	플래시 5
문법	<i>theNode.nodeName</i>
액세스	읽기/쓰기

설명

nodeName은 theNode의 이름을 나타내는 문자열 속성이다. 액션스크립트에서 지원되는 노드의 유형은 두 가지(원소 노드와 텍스트 노드)뿐이므로, nodeName 값은 다음과 같은 식으로 결정된다.

- theNode가 원소 노드이면 nodeName은 그 원소의 태그 이름과 같은 문자열이 된다. 예를 들어 theNode가 <BOOK> 원소를 나타낸다면 theNode.nodeName은 "BOOK"이 된다.
- theNode가 텍스트 노드이면 nodeName은 null이 된다. 이러한 규칙은 텍스트 노드에 대한 nodeName은 "#text"라는 문자열이어야 한다는 DOM 스펙과는 다르다. 원한다면 DOM과 호환되는 nodeType 속성을 이용할 수도 있다.

예제

nodeName을 이용하여 현재 노드가 원하는 원소 유형에 해당하는지 확인할 수 있다. 예를 들어 아래 예제에서는 XML 문서의 첫 번째 레벨에 있는 H1 태그의 모든 내용을 추출한다(이 예제에서는 소문자로 표시된 h1 태그는 확인할 수 없고 대문자로 표시된 H1 태그만 확인할 수 있다).

```
myDoc = new XML('<H1>first heading</H1><P>content</P>' +
                '<H1>second heading</H1><P>content</P>');
for (i = 0; i < myDoc.childNodes.length; i++) {
    if (myDoc.childNodes[i].nodeName == "H1") {
        trace(myDoc.childNodes[i].firstChild.nodeValue);
    }
}
```

참조

XML.nodeType, XML.nodeValue

XML.nodeType 속성

현재 노드의 유형

버전 플래시 5**문법** *theNode.nodeType***액세스** 읽기 전용**설명**

nodeType은 theNode의 유형을 리턴하는 정수 속성이다. 액션스크립트에서는 두 가지 유형의 노드(원소 노드와 텍스트 노드)만 지원되기 때문에, nodeName은 두 종류의 값만을 가질 수 있다(원소 노드는 1, 텍스트 노드는 3). 이러한 값은 얼핏 보기에는 별 뜻 없이 정해진 것 같지만, 사실 DOM에서 지정한 값을 따른 것이다. 참고로 다른 DOM의 노드 유형은 [표 R-13]에 나온 것과 같다.

[표 R-13] DOM 노드 유형

노드 종류	노드 유형 코드
ELEMENT_NODE*	1
ATTRIBUTE_NODE	2
TEXT_NODE*	3
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11
NOTATION_NODE	12

* 플래시에서 지원되는 유형. 정확하게 말하면 액션스크립트에서는 원소(element) 노드와 텍스트(text) 노드 외에도 attribute, document, document_type 노드를 지원하지만, 그러한 노드를 객체 형태로 직접 액세스할 수 없다. 예를 들어 attributes 속성을 통해 노드의 특성을 조작할 수는 있지만 attribute 노드 자체를 직접 액세스할 수는 없다. 마찬가지로 docTypeDecl 속성을 통해 문서의 DOCTYPE 태그를 액세스할 수는 있지만, document_type 노드 자체를 직접 액세스할 수는 없다.

원소 노드는 XML이나 HTML 태그에 해당한다. 예를 들어 `<P>what is your favorite color?</P>`라는 XML 코드에서 P 태그는 XML 객체 계층에서 원소 노드(`nodeType`이 1)로 표현된다. XML 소스 코드에 있는 태그에 포함되는 텍스트(예: “what is your favorite color?”)는 텍스트 노드(`nodeType`이 3)로 표시된다.

예제

`nodeType` 값에 따라 조건적으로 노드를 처리하는 경우도 있다. 예를 들어 다음 예제에서는 `theNode`의 자식 중에서 비어있는 텍스트 노드를 모두 제거한다.

```
// theNode의 모든 자식에 대해 루프를 돌린다.
for (i = 0; i < theNode.childNodes.length; i++) {
    // 현재 노드가 텍스트 노드이면
    if (theNode.childNodes[i].nodeType == 3) {
        // 노드에 공백이 아닌 문자가 있는지 확인한다.
        var j = 0;
        var emptyNode = true;
        for (j = 0; j < theNode.childNodes[i].nodeValue.length; j++) {
            // 공백이 아닌 문자는 ASCII 값이 32보다 커야 한다.
            if (theNode.childNodes[i].nodeValue.charCodeAt(j) > 32) {
                emptyNode = false;
                break;
            }
        }
        // 공백문자만으로 이루어진 경우에는 그 노드를 삭제한다.
        theNode.childNodes[i].removeNode();
    }
}
```

참조

XML 클래스, `XML.nodeName`, `XML.nodeValue`

XML.nodeValue 속성

현재 노드의 값

버전	플래시 5
문법	<code>theNode.nodeValue</code>
액세스	읽기/쓰기

설명

nodeValue는 theNode를 문자열로 표현한 값을 나타내는 속성이다. 액션스크립트에서 지원하는 노드의 유형은 두 가지(원소 노드와 텍스트 노드)뿐이므로 nodeValue의 값은 두 종류로 나뉜다.

- theNode가 원소 노드이면 nodeValue는 null이 된다.
- theNode가 텍스트 노드이면 nodeValue는 그 노드에 들어있는 텍스트가 된다.

일반적으로 nodeValue는 텍스트 노드에서만 사용한다. 텍스트 노드에 새로운 텍스트를 대입할 때는 다음과 같은 식으로 nodeValue를 활용한다.

```
// 새로운 XML 문서를 만든다.
myDoc = new XML('<H1>first heading</H1><P>content</P>');

// H1 태그에 들어있는 텍스트를 바꾼다.
myDoc.firstChild.firstChild.nodeValue = "My Life Story";
```

nodeValue를 이용하여 직접 텍스트 노드의 값을 구할 수도 있지만, 문자열이 들어갈 자리에 텍스트 노드를 사용하면 자동으로 toString() 메소드가 호출되므로 굳이 사용하지 않아도 된다. 다음과 같은 구문을 사용하면 텍스트 노드의 텍스트가 Output 창에 출력된다.

```
trace(myDoc.firstChild.firstChild);
```

참조

XML.nodeName, XML.nodeType

XML.onData() 이벤트 핸들러

외부 XML 소스 코드를 불러오는 작업이 완료되었을 때(파싱하기 전에) 실행되는 이벤트 핸들러

버전 플래시 5(문서에는 나와있지 않음)

문법 `XMLdoc.onData(src);`

인자

src 로딩된 XML 소스 코드를 포함하는 문자열

설명

onData() 핸들러는 load()나 sendAndLoad()를 호출한 결과로 XML 소스 코드를 로딩하는 작업이 완료되었을 때 자동으로 실행된다. onData()는 다음과 같은 식으로 동작한다.

- 불러온 소스 코드가 undefined이면 XMLdoc.onLoad()를 호출하는데, 이 때 success 매개변수를 false로 설정한다.
- 그렇지 않으면 불러온 소스를 파싱하여 XMLdoc에 저장하고 XMLdoc.loaded를 true로 설정한 다음 XMLdoc.onLoad()를 호출하는데, 이 때 success 매개변수는 true로 설정한다.

onData() 핸들러에 사용자가 정의한 콜백 함수를 대입하여 액션스크립트에서 파싱하기 전에 XML 소스 코드를 가로챌 수도 있다. 경우에 따라 직접 XML 소스를 조작하는 것이 액션스크립트에 내장된 파싱 기능을 이용하는 것보다 더 빠를 수도 있다.

예제

아래 예제에서는 XML 소스 코드를 불러와서 액션스크립트에서 파싱하지 않고 바로 Output 창으로 출력하는 방법을 보여준다.

```
myDoc = new XML();
myDoc.onData = function (src) {
    trace("Here's the source: \n" + src);
};
myDoc.load("book.XML");
```

참조

XML.onLoad()

XML.onLoad() 이벤트 핸들러

외부 XML 데이터를 로딩하고 파싱하는 작업이
완료되었을 때 실행되는 핸들러

버전 플래시 5

문법 `XMLdoc.onLoad(success)`

인자

success 로딩 작업이 제대로 처리되었는지(true) 아닌지(false)를 나타내는 부
울 값

설명

XMLdoc의 onLoad() 이벤트 핸들러는 load()나 sendAndLoad()를 이용하여 외부 XML 파일을 로딩했을 때 실행된다. 기본적으로 XML 문서 객체의 onLoad() 핸들러는 비어있는 함수이다. onLoad()를 이용하려면 다음과 같은 식으로 콜백 핸들러(즉 사용자 정의 함수)를 대입해야 한다.

```
myDoc = new XML();
myDoc.onLoad = handleLoad;
function handleLoad (success) {
    // XML을 처리하는 부분
}
```

XMLdoc을 처리해도 되는지 알아볼 때는 onLoad()를 이용한다. onLoad()가 실행되면 외부 XML 데이터를 로딩하고 파싱하는 과정이 완료되었음을 알 수 있기 때문에, 로딩된 내용을 안전하게 액세스할 수 있다. 따라서 onLoad() 핸들러를 이용하면 XML load() 함수를 호출한 다음 데이터가 모두 전송될 때까지 프리로더 코드를 별도로 만들지 않아도 된다. 예를 들어 아래 코드에서는 XML 문서 로딩을 시작하고 나서 로딩이 완료되면 자동으로 handleLoad() 함수가 실행될 때까지 기다리면 된다. 로딩이 성공적으로 완료되면 displayProduct() 함수를 이용하여 XML 내용을 처리한다. 그렇지 않으면 displayError() 함수를 이용하여 오류 메시지를 출력한다(아래 코드에서는 생략했지만 displayProduct()와 displayError()는 각각 로딩이 성공했을 때와 실패했을 때 적절한 메시지를 출력하기 위한 함수라고 생각하면 된다).


```
myDoc = new XML();
myDoc.onLoad = handleLoad;
myDoc.load("productInfo.XML");

function handleLoad(success) {
    if (success) {
        output = "Product information received";
        displayProduct(); // 사용자 정의 출력 함수를 호출한다.
    } else {
        output = "Attempt to load XML data failed";
        displayError(); // 사용자 정의 출력 함수를 호출한다.
    }
}
```

handleLoad()의 success 인자의 값은 로딩 처리 결과에 따라 인터프리터에서 자동으로 true 또는 false로 설정한다. 하지만 success 인자는 실질적으로 그다지 유용한 것은 아니다. 대부분의 웹 서버에서는 오류 메시지("404 File Not Found"와 같은 메시지)를 HTML 문서 형태로 전송한다. HTML은 XML 데이터 형태로 파싱할 수 있기 때문에, 서버 오류 페이지 결과를 받으면 그 페이지는 XML 문서 객체로 파싱할 수 있다. 오류 메시지를 파싱하는 작업에서는 문제가 생기지 않기 때문에, 서버에서 오류가 발생한 경우에도 로딩 작업이 제대로 처리된 것으로 간주되고 success 인자는 true가 된다. 실제 요청한 데이터를 받았는지 확인하려면 받은 데이터의 구조를 테스트해보거나 특정한 자식의 nodeName을 확인하는 것과 같이 데이터의 특징을 직접 확인하는 방법을 사용하는 것이 좋다. 사용자 정의 파싱을 하는 데 사용할 수 있는 XML.onData() 이벤트 핸들러 부분도 참조해 보자.

참조

XML.load(), XML.onData(), XML.sendAndLoad()

XML.parentNode 속성

주어진 노드의 부모 노드에 대한 레퍼런스

버전	플래시 5
문법	<i>theNode</i> .parentNode
액세스	읽기 전용

설명

parentNode는 XML 객체 계층에서 theNode의 부모 노드에 해당하는 노드 객체에 대한 레퍼런스를 리턴하는 속성이다. theNode가 현재 계층의 최상위 노드라면 parentNode에서 null을 리턴한다.

다음과 같은 XML 소스에서 MESSAGE 노드는 “hey” 텍스트 노드의 parentNode가 된다.

```
<MESSAGE>hey</MESSAGE>
```

아래 소스에서 ROOM 노드의 parentNode는 HOTEL 노드이다.

```
<HOTEL><ROOM><SIZE>Double</SIZE></ROOM></HOTEL>
```

참조

XML.childNodes, XML.firstChild, XML.lastChild, XML.previousSibling

XML.parseXML() 메소드

XML 소스 코드 문자열을 파싱하는 메소드

버전	플래시 5
문법	<code>XMLdoc.parseXML(string)</code>
인자	
<i>string</i>	XML 소스 코드로 이루어진 문자열

설명

parseXML() 메소드는 내부의 load() 함수와 비슷하다. 이 메소드는 string에 포함된 XML 소스를 읽어서 파싱하고 그 XML을 객체 계층으로 변환한 다음, 그 결과를 XMLdoc에 집어넣는다. 이전에 XMLdoc에 들어있던 내용은 새로운 객체로 대체된다. XMLdoc은 XMLnode가 아닌 XML 클래스의 객체라는 점에 주의하자.

텍스트 노드에 HTML이나 XML 소스 코드를 파싱하지 않은 채로 저장하고 싶다면 다음과 같이 CDATA를 사용하면 된다.

```
<![CDATA[ source ]]>
```

예를 들어 다음 예제에서는 MESSAGE 원소를 만드는데, 이 원소에는 “Welcome to my site”(이 때 태그는 XML 태그로 간주되지 않으며 따라서 XML 객체 계층에 들어가지도 않는다)라는 텍스트 노드가 자식 노드로 포함된다.

```
myDoc = new XML();
myDoc.parseXML("<MESSAGE><![CDATA[<B>Welcome</B> to my site]]>
    </MESSAGE>");
trace(myDoc); // "<MESSAGE><B>Welcome</B> to my site</MESSAGE>"
```

예제

parseXML()은 XML 객체에 있는 현재 객체 계층을 내부에서 만든 XML 소스 코드(예를 들면 사용자가 입력한 내용)를 기반으로 새로운 계층으로 바꾸기 위한 용도로 사용된다. 아래 예제에서는 username과 content라는 입력 텍스트 필드와 마크업을 결합시켜 간단한 XML 메시지를 만든다.

```
myDoc = new XML();
myXMLsource = "<MESSAGE><USER>" + username + "</USER><CONTENT>"
    + content + "</CONTENT></MESSAGE>";
myDoc.parseXML(myXMLsource);
```

참조

XML.load(), XML.status

XML.previousSibling 속성

주어진 노드 바로 앞에 있는 노드에 대한 레퍼런스

버전	플래시 5
문법	theNode.previousSibling
액세스	읽기 전용

설명

previousSibling은 XML 객체 계층의 현재 레벨에서 theNode 바로 앞에 있는 노드 객체에 대한 레퍼런스를 리턴하는 속성이다. 현재 레벨의 theNode 앞에 다른 노드가 없다면 null을 리턴한다.

다음 XML 소스에서 CONTENT 노드의 previousSibling은 USER 노드이다.

```
<MESSAGE><USER>gray</USER><CONTENT>hi</CONTENT></MESSAGE>
```

예제

previousSibling 속성은 XML 객체 계층을 횡단하기 위한 용도로 쓰일 수 있다 (하지만 이러한 용도로는 nextSibling을 더 많이 사용한다). theNode의 모든 자식들을 순서를 뒤집어서 출력하고 싶다면 다음과 같은 코드를 사용하면 된다.

```
for (var i = theNode.lastChild; i != null; i = i.previousSibling) {
    trace("found node: " + i.nodeName);
}
```

참조

XML.childNodes, XML.firstChild, XML.lastChild, XML.nextSibling, XML.nodeName, XML.nodeValue, XML.parentNode

XML.removeNode() 메소드

XML 객체 계층에서 노드를 삭제하는 메소드

버전 플래시 5

문법 theNode.removeNode()

설명

removeNode()는 XML 문서에서 theNode를 삭제하는 메소드이다. 이 때 theNode의 모든 자손(자식, 손자,...)도 함께 지워진다. theNode의 부모의 childNodes 속성도 자동으로 갱신되어 새로운 객체 계층의 구조가 기록된다.

예제

아래 예제에서는 두 번째 자식 노드를 삭제한다. 세 번째 자식 노드가 두 번째 자식 노드의 자리를 차지하게 된다.

```
myDoc = new XML("<P>one</P><P>two</P><P>three</P>");
myDoc.childNodes[1].removeNode();
trace(myDoc); // "<P>one</P><P>three</P>"가 출력된다.
```

참조

XML.appendChild()

XML.send() 메소드

XML 소스 코드를 스크립트로 전송하는 메소드

버전 플래시 5**문법** *XMLdoc.send(URL, window)***인자****URL** XMLdoc을 보낼 스크립트 또는 애플리케이션 위치를 나타내는 문자열**window** 스크립트에서 보낸 결과를 출력할 브라우저 창 또는 프레임의 이름을 나타내는 문자열. 사용자가 정한 이름을 사용해도 되고 미리 정의된 이름인 “_blank”, “_parent”, “_self”, “_top” 중 하나를 사용해도 된다. 자세한 내용은 getURL() 전역 함수 항목에 나온 창 설정 부분을 참조하기 바란다.**설명**

send()는 XMLdoc을 XML 소스 코드 문자열로 변환하여 그 코드를 URL 위치에 있는 스크립트나 애플리케이션으로 HTTP 요청 형태로 전송하는 메소드이다. URL 위치에 있는 스크립트 또는 애플리케이션에서 XML 코드를 처리하고 상황에 따라 어떤 응답(보통 웹 페이지 형태)을 브라우저로 보내게 되며, 그 내용은 window 인자로 지정한 창으로 출력된다. 이 때 응답을 플래시가 아닌 브라우저에서 받아들이는 점에 주의하자. 플래시에서 그 응답을 처리하려면 sendAndLoad() 메소드를 사용해야 한다.

브라우저에서 실행 중인 플래시 플레이어에서 XML.send()를 호출하면 XMLdoc은 POST 메소드를 통해 전송된다. XML.send()를 독립적으로 실행되는 플래시 플레이어에서 실행하면 XMLdoc이 GET 메소드를 통해 전송된다. 이 XML 문자열을 받아들이는 서버 애플리케이션은 HTTP 요청으로 전송된 POST 데이터를 일반적인 이름/값 쌍으로 파싱하지 않고 그대로 액세스할 수 있어야 한다. 펄에서는 POST 요청으로 들어온 데이터를 STDIN을 통해 처리할 수 있기 때문에, 다음과 같이 \$buffer 같은 변수에 저장할 수 있다.

```
read(STDIN,$buffer,$ENV{'CONTENT_LENGTH'});
```

ASP에서는 POST로 받은 데이터를 Request.BinaryRead 메소드를 이용하여 처리할 수 있다. 애플리케이션 중에는 POST를 통해 받은 데이터를 바로 처리할 수 없는 것도 있다(예: 콜드퓨전). 이런 경우에는 XML.toString()을 이용하여 XML 객체를 문자열로 변환한 다음에 loadVariables를 이용하여 문자열을 변수 형태로 전송해야 한다.

서버로 보내는 XML 텍스트의 기본 MIME 유형은 application/x-www-urlform-encoded이다. 하지만 텍스트 자체가 URL 인코딩된 것은 아니다. 플래시 5 빌드 41 이후 버전에서는 XML.contentType 속성을 이용하여 MIME 유형을 바꿀 수 있다. 예를 들어 다음과 같은 코드를 사용하면 MIME 유형을 application/XML로 바꿀 수 있다.

```
myXML = new XML();
myXML.contentType = "application/XML";
```

물론 contentType 속성을 application/x-www-urlform-encoded로 지정한다고 해서 전송되는 텍스트가 URL 인코딩 처리되지 않는다.

플래시 5 빌드 41부터는 XML 소스를 파싱할 때 텍스트 노드에 &, ', ", <, > 기호가 나타나면 자동으로 &, ', ", >, <로 변환된다. XML 객체를 다시 문자열로 변환할 때는 원래 기호로 다시 변환되기 때문에 따로 신경쓸 필요는 없다. 하지만 XML 소스를 서버로 전송할 때는 이렇게 변환된 형태로 전송된다.

예제

```
myDoc = new XML("<SEARCH_TERM>tutorials</SEARCH_TERM>");
myDoc.send("http://www.domain.com/cgi-bin/lookup.cgi", "remoteWin");
```

참조

XML.sendAndLoad(), XML.load(), XML.loaded, XML.onLoad(), XML.status

XML.sendAndLoad() 메소드

XML 소스 코드를 스크립트로 전송하고
XML 소스를 받기 위한 메소드

버전	플래시 5
문법	<code>XMLdoc.sendAndLoad(URL, resultXML)</code>
인자	
URL	XMLdoc을 보낼 스크립트나 애플리케이션의 위치를 나타내는 문자열
resultXML	서버에서 리턴하는 XML 소스 코드를 받아들이 XML 문서 객체에 대한 레퍼런스

설명

`sendAndLoad()`는 XMLdoc을 XML 소스 코드 문자열로 직렬화하고 나서 그 코드를 URL 위치에 있는 스크립트나 애플리케이션으로 전송하는 메소드이다. 스크립트 또는 애플리케이션에서는 전달받은 XML을 처리하고 그 결과로 어떤 XML 문서를 다시 플래시로 전송한다. 플래시에서는 이렇게 전송받은 문서를 파싱하고 XML 객체 계층으로 변환하여 `resultXML`에 저장한다. 원래 `resultXML`에 들어있던 내용은 모두 삭제되고 새로 만들어진 XML 내용이 그 자리를 차지하게 된다. XML을 서버로 전송하는 것과 관련된 정보는 `XML.send()` 항목을 참조하기 바란다.

주의 사항

`sendAndLoad()` 메소드를 이용하여 가져온 내용을 액세스하기 전에 로딩 및 파싱 작업이 완료되었는지 확인해야 한다. 보통 로딩 및 파싱이 완료되었는지 확인할 때는 `resultXML`의 `loaded` 속성 값을 체크하거나 `resultXML`에 `onLoad()` 이벤트 핸들러를 대입하여 로딩이 완료되면 필요한 작업을 처리하도록 만드는 방법을 사용한다. 자세한 내용은 `XML.loaded`와 `XML.onLoad()` 항목에서 찾을 수 있다. 가져온 데이터를 파싱한 결과를 확인할 때는 문서의 `status` 속성을 체크하면 된다.

`XML.sendAndLoad()`는 `loadVariables()` 전역 함수 항목에 나온 [표 R-8]에 수록된 도메인 기반의 보안 규제를 따른다.

예제

```
// XML 문서를 만든다.
myDoc = new XML("<P>hello server!</P>");
```

```
// 서버의 응답을 받아들이기 위한 비어있는 XML 문서를 만든다.
serverResponse = new XML();

// myDoc을 서버로 보내고 그 결과로 받은 XML은 serverResponse에 저장한다.
myDoc.sendAndLoad("http://www.domain.com/cgi-bin/readData.cgi",
    serverResponse);

// serverResponse에 onLoad() 이벤트 핸들러를 대입하여 서버로부터
// 받은 결과를 output이라는 텍스트 필드에 출력한다.
serverResponse.onLoad = function () {
    output = serverResponse.toString();
}
```

서버와 XML을 주고받는 것과 관련된 기본적인 내용은 다음 URL에 있는 매크로 미디어의 문서에 자세히 설명되어 있다.

<http://www.macromedia.com/support/flash/interactivity/XML>

참조

XML.load(), XML.loaded, XML.onLoad(), XML.send(), XML.status

XML.status 속성

XML 소스를 객체 계층으로 파싱하는 작업이 제대로 처리되었는지를 나타내는 속성

버전	플래시 5
문법	XMLdoc.status
액세스	읽기 전용

설명

status는 XML 소스 코드를 파싱하는 과정에서 오류가 발생했는지 여부를 나타내는 상태 코드를 숫자 형태로 리턴하는 속성이다. 파싱은 다음과 같은 경우에 이루어진다.

- XML 소스를 XML() 생성자의 인자로 전달했을 때
- parseXML() 메소드를 호출했을 때

- load()나 sendAndLoad() 메소드를 이용하여 XML 소스를 새로운 XML 객체로 불러왔을 때

status 속성에 저장되는 상태 코드는 [표 R-14]에 나와 있다. 파싱할 때 오류가 하나도 없었다면 status가 0이 된다. 오류가 발생하면 status 값이 음수가 된다. 오류가 한 번이라도 발생하면 파싱 작업이 중지되기 때문에, 오류가 생긴 부분을 수정한 다음에도 그 뒤에 있는 오류 때문에 다시 오류가 발생할 수 있다.

[표 R-14] XML 파싱 상태 코드

상태	설명
0	문서를 파싱할 때 오류가 발생하지 않은 경우(파싱 성공)
-2	CDATA 섹션을 제대로 마치지 않은 경우
-3	XML 선언을 제대로 마치지 않은 경우
-4	DOCTYPE 선언을 제대로 마치지 않은 경우
-5	주석을 제대로 마치지 않은 경우
-6	XML 원소의 형식이 틀린 경우
-7	XML 소스를 파싱하기 위한 메모리가 불충분한 경우
-8	특성 값을 제대로 끝마치지 않은 경우
-9	시작 태그에 해당하는 종료 태그가 없는 경우
-10	종료 태그에 해당하는 시작 태그가 없는 경우

status는 외부에서 불러온 XML 파일을 처리해도 될지 결정하기 위한 용도로 쓰인다. status를 체크하기 전에 loaded 속성을 미리 확인하여 load()나 sendAndLoad() 메소드가 제대로 실행되었는지 알아보는 것이 좋다. 액션스크립트의 XML 파서에서는 DTD를 기준으로 문서를 검증하지 않고 XML 형식을 제대로 갖추고 있는지 여부만 확인한다.

예제

```
myDoc = new XML("<BOOK>Colin Moock</AUTHOR></BOOK>");
trace(myDoc.status); // "-10"이 출력된다. (시작 태그가 없음)
```

참조

XML.load(), XML.loaded, XML.onLoad(), XML.parseXML(), XML.sendAndLoad()

XML.toString() 메소드

XML 노드의 소스 코드를 문자열로 변환한 값을 리턴하는 메소드

버전 플래시 5

문법 `theNode.toString()`

리턴 값

theNode에서 시작하는 XML 객체 계층의 소스 코드를 나타내는 문자열

설명

toString()은 XML 노드 객체나 XML 문서 객체를 XML 소스 코드로 변환하는 메소드이다. theNode가 XML 문서 객체의 최상위 레벨에 해당하면 DOCTYPE과 XML 선언 태그도 문자열에 포함된다. 문서의 ignoreWhite 속성이 false이면 공백도 그대로 유지되며 문서의 소스 코드가 처음 파싱할 때의 모습 그대로 다시 복구된다.

일반적으로 toString() 메소드를 직접 사용하는 경우는 거의 없다. 문자열이 들어갈 자리에 theNode를 사용하면 액션스크립트에서 자동으로 toString() 메소드를 호출하기 때문이다.

예제

```
var myDoc = new XML('<?XML version="1.0"?><!DOCTYPE foo SYSTEM "bar.dtd">
    <BOOK>
    <TITLE>ActionScript: The Definitive Guide</TITLE>'
    + '<AUTHOR SALUTATION="Mr">Colin Moock </AUTHOR>    '
    + '<PUBLISHER>O\'reilly & Associates, Inc</PUBLISHER>    </BOOK>');

trace(myDoc.toString());
// 출력되는 내용
<?XML version="1.0"?><!DOCTYPE foo SYSTEM "bar.dtd">
<BOOK>    <TITLE>ActionScript:
    The Definitive Guide</TITLE><AUTHOR SALUTATION="Mr">Colin
Moock </AUTHOR>    <PUBLISHER>O'reilly & Associates, Inc
</PUBLISHER>    </BOOK>
```

참조

Object.toString(), XML.nodeValue

XML.XMLDecl 속성

문서의 XML 선언 태그

버전	플래시 5
문법	<code>XMLdoc.XMLDecl</code>
액세스	읽기/쓰기

설명

XMLDecl은 XMLdoc의 XML 선언부를 나타내는 문자열 속성이다. 만약 XML 선언부가 없다면 XMLDecl 값은 null이 된다. 이 때 XMLdoc은 XML 객체 계층의 최상위 레벨에 있는 노드라는 점에 주의하자(즉 XmlNode가 아닌 XML 클래스의 인스턴스이다).

XML 문서의 XML 선언 태그는 보통 그 문서에서 사용하는 XML 버전을 나타내기 위한 용도로 쓰인다. 외부에서 검증받을 수 있는 제대로 된 형식을 갖춘 XML 문서를 만들 때는 XML 선언 태그를 사용해야 한다.

예제

```
// 제대로 된 형식을 갖춘 문서
// (액션스크립트에서는 DTD와 비교하여 검증하지 않음)
myXML = new XML('<?XML version="1.0"?><P>this is a short document</P>');
trace(myXML.XMLDecl); // "<?XML version="1.0"?>"가 출력됨
// XML 선언을 새로 설정한다.
myXML.XMLDecl = '<?XML version="1.0" standalone="no"?>';
```

참조

XML.docTypeDecl

XMLnode 클래스

XML 클래스의 내부 수퍼클래스

버전 플래시 5

설명

XMLnode는 XML 객체 계층에 있는 노드의 핵심적인 속성과 메소드를 정의하는 클래스이다. XMLnode는 내부적인 도구이지만 XML 객체의 기본 기능을 확장하기 위해 프로그래머가 직접 이 클래스를 사용할 수도 있다.

모든 XML 객체 계층에는 다음과 같은 두 가지 객체 노드가 포함된다.

- 계층에 필요한 메인 컨테이너 역할을 하는 XML 노드
- 메인 컨테이너 노드의 자식 노드에 해당하는 여러 개의 XMLnode 노드

메인 컨테이너 노드는 XML 클래스의 인스턴스이다. 예를 들어 다음과 같이 myDoc 객체를 만들면

```
myDoc = new XML();
```

myDoc은 XML 클래스의 인스턴스가 된다. XML 클래스는 XMLnode로부터 파생된 노드이므로 메인 컨테이너 노드에는 XMLnode 및 XML 클래스에서 정의된 모든 속성과 메소드가 포함된다. 하지만 myDoc의 자식은 XML 클래스가 아닌 XMLnode 클래스의 인스턴스가 된다.

따라서 다음과 같은 코드를 이용하여 myParagraph를 만들면

```
myParagraph = myDoc.createElement("P");
```

myParagraph는 XMLnode 클래스의 인스턴스가 된다. 대부분의 경우에 노드 클래스 사이의 내부적인 차이는 XML 객체에 영향을 미치지 않는다. 하지만 모든 XML 객체에 상속되는 속성이나 메소드를 새로 추가하고 싶다면 XML이 아닌 XMLnode 클래스의 prototype 속성을 사용해야 한다(뒤에 나오는 예제 참조). XMLnode.prototype에 들어있는 메소드와 속성은 무비에 있는 모든 XML 노드에 상속된다.

독자들의 편의를 위해 XMLnode와 XML 클래스에서 정의된 속성, 메소드, 이벤트 핸들러를 [표 R-15]에 정리해 놓았다. XML 클래스의 인스턴스에서는 이 표에 나온 모든 아이템을 사용할 수 있지만, XML 클래스에서 정의된 아이템은 XMLnode의

인스턴스에서는 사용할 수 없다. 예를 들어 load() 메소드는 XML 클래스의 인스턴스에서는 호출할 수 있지만 XMLnode 클래스의 인스턴스에서는 호출할 수 없다. 각 아이টে에 대한 자세한 설명은 XML 클래스의 해당 아이টে 항목을 참조하기 바란다.

[표 R-15] XMLnode와 XML 클래스의 속성, 메소드 및 이벤트 핸들러 목록

XMLnode와 XML	XML 전용
appendChild()	contentType
attributes	createElement()
childNodes	createTextNode()
cloneNode()	docTypeDecl
firstChild	ignoreWhite
hasChildNodes()	load()
insertBefore()	loaded
lastChild	onData()
nextSibling	onLoad()
nodeName	parseXML()
nodeType	send()
nodeValue	sendAndLoad()
parentNode	status
previousSibling	xmlDecl
removeNode()	
toString()	

예제

다음과 같은 코드를 이용하면 XMLnode.prototype에 사용자가 직접 만든 secondChild()라는 메소드를 추가할 수 있다(이렇게 하면 무비에 있는 모든 XML 노트에서 secondChild() 메소드를 사용할 수 있다).

```
XMLnode.prototype.secondChild = function () {
    return this.childNodes[1];
};

myDoc = new XML("<PRODUCT>Cell Phone</PRODUCT><PRODUCT>Game Console</PRODUCT>");

trace(myDoc.secondChild()); // "<PRODUCT>Game Console</PRODUCT>"
```

XML.prototype을 이용하여 XML 클래스를 확장해도 되지만 그렇게 하면 메인 컨테이너 노드(즉 XML 클래스의 인스턴스)에서만 그 아이템을 사용할 수 있다.

참조

XML 클래스, 12장의 '수퍼클래스와 서브클래스'

XMLSocket 클래스

연속적인 서버/클라이언트 TCP/IP 연결을
지원하기 위한 클래스

버전 플래시 5

생성자 new XMLSocket ()

메소드

close() 서버 애플리케이션 접속을 종료하는 메소드

connect() 서버 애플리케이션에 접속하는 메소드

send() XML 객체 계층을 문자열 형태로 서버 애플리케이션에 전달하는 메소드

이벤트 핸들러

onClose() 서버와의 접속이 끝났을 때 실행되는 이벤트 핸들러

onConnect() 서버와 연결되었을 때 실행되는 이벤트 핸들러

onData() 데이터를 받았을 때(XML 파싱 이전에) 실행되는 이벤트 핸들러

onXML() 데이터를 받아서 XML 객체 계층으로 파싱하는 작업이 끝났을 때 실행되는 이벤트 핸들러

설명

플래시와 서버 사이의 연결은 대부분 짧은 시간 안에 끝난다. 플래시에서 loadMovie(), loadVariables(), XML.load() 함수를 이용하여 외부 데이터를 요청하면 임시 통신 채널이 생성된다. 데이터는 그 채널을 통해 전송되고 작업이 끝나면 통신 채널도 종료된다. 이와 같은 짧은 시간 안에 끝나는 통신은 매우 유용하지만 다음과 같은 두 가지 측면에서 제약을 받게 된다.

- 연결이 닫히고 나면 서버에서 플래시에 연락할 수 없다. 언제나 플래시쪽에 서만 서버에 연결할 수 있기 때문이다.
- 플래시에서 서버의 정보를 가져올 때마다 새로 접속해야 한다. 계속해서 새로 접속하다 보면 시간도 많이 걸리고 CPU에도 부담이 되기 때문에, 플래시에서 서버와 실시간 트랜잭션을 처리하기가 어렵다.

플래시 5 이후로는 XMLSocket 클래스를 이용하면 서버 애플리케이션과 플래시 사이에 지속적인 통신 채널을 열 수 있기 때문에, 이러한 제약을 극복할 수 있다. XMLSocket은 서버 대화방이나 네트워크 게임과 같이 자주 서버 데이터를 업데이트 해야 하는 프로그램을 만들기 위한 용도로 쓰인다.

XMLSocket을 이용하여 원격 애플리케이션에 연결하려면, 우선 다음과 같이 XMLSocket 객체를 만들어서 변수에 저장해야 한다.

```
mySocket = new XMLSocket();
```

그리고 나서 connect() 메소드를 호출하여 플래시에서 원격 애플리케이션과 통신 채널을 연결하도록 만든다. 예를 들어 다음과 같이 하면 된다.

```
mySocket.connect("http://www.myserver.com", 8000);
```

일단 연결이 성사되면 XMLSocket 객체가 송수신기 역할을 맡는다. 소켓의 send() 메소드를 호출하여 XML 형식의 데이터를 원격 애플리케이션으로 보낼 수 있고 onXML() 이벤트를 통해 소켓에서 XML 형식의 데이터를 받았다는 것을 알 수 있다.

XMLSocket 객체와 함께 사용할 서버 애플리케이션은 다음과 같은 조건을 만족시켜야 한다.

- 1024번 이상의 포트를 통해 TCP/IP 소켓 연결을 할 수 있어야 한다.
- 제로 바이트(즉 ASCII의 null 문자)로 구분되는 세그먼트로 XML 형식의 데이터를 전송해야 한다.

서버 애플리케이션은 보통 플래시 프로그래머가 만드는 것이 아니라 서버를 담당하는 프로그래머가 만든다. 온라인 코드 창고에 있는 Java XMLSocket server 프로그램은 서버 애플리케이션에서 받은 모든 메시지를 연결된 모든 클라이언트로 보내는 간단한 예이다.

XMLSocket 연결은 다음 중 한 가지 상황이 일어나기 전까지 지속된다.

- XMLSocket 객체의 close() 메소드가 호출되는 경우
- XMLSocket 객체에 대한 레퍼런스가 더 이상 없는 경우
- 서버에서 연결을 중단시키는 경우(이렇게 하면 onClose() 이벤트가 발생한다)
- 무비를 닫거나 플래시 플레이어を終료시키는 경우

XMLSocket 클래스에서는 연결 상태를 확인할 수 있는 방법도 제공한다. XML Socket에는 onClose, onConnect, onXML이라는 세 가지 속성이 들어있는데, 이 속성을 이용하여 각 상황에 따른 이벤트 핸들러를 정의할 수 있다. 이러한 핸들러는 프로그래머가 직접 제어할 수 없는 이벤트에 의해 플래시에서 자동으로 호출하기 때문에, 보통 '콜백 핸들러(callback handler)'라고 부른다(이런 면에서 이 속성은 액션스크립트에 내장된 클립 및 버튼 이벤트 핸들러와 매우 유사하지만 핸들러 함수는 사용자가 정의한다는 점에서 차이가 난다). 예를 들어 서버에서 연결을 끊으면 onClose 속성으로 정의된 핸들러가 실행된다.



onClose와 onConnect 속성에 대한 콜백 핸들러를 정의하지 않으면 오류를 체크하거나 피드백을 처리할 수 없다. onXML 속성에 대한 콜백 핸들러를 정의하지 않으면 서버측 애플리케이션에서 보낸 데이터가 소켓에서 도착하더라도 전혀 알 수 없고 그 데이터의 내용도 알아낼 수 없다.

예제

다음 예제는 간단한 채팅 클라이언트를 만들기 위한 기본 골격을 구현한 코드이다. 다음 사이트에서 클라이언트를 실제 실행시킨 모습을 확인할 수 있다.

<http://www.moock.org/chat>

서버와 클라이언트는 모두 온라인 코드 창고에서 구할 수 있다.

```
// 간단한 채팅 클라이언트
// *** 일반적인 초기화 부분
var incomingUpdated = false; // incoming (메인 채팅 텍스트 필드)의
// 끝까지 스크롤해야 할지 추적하기 위한 변수
var incoming = ""; // 메인 채팅 텍스트 필드에 초기 값을 대입한다.
```



```
// 스크롤을 관리하기 위한 무비를 추가한다. 새로운 메시지가 추가될 때마다
// 텍스트 필드에 다음(가장 최근)에 입력된 줄이 출력되도록 만든다.
// 스크롤 관리자가 필요한 이유는 플래시 5 플레이어의 빌드 30에 있는
// 텍스트 필드 스크롤과 관련된 버그 때문이다.
attachMovie("processScroll", "processScroll", 0);

// 메시지를 받았을 때 재생할 사운드를 추가한다.
var click = new Sound();
click.attachSound("click");

// 사용자가 대화방에 참여하거나 대화방을 떠날 때 재생할 사운드를 추가한다.
var welcomeSound = new Sound();
welcomeSound.attachSound("welcome");

// 버튼에 있는 노란색 하이라이트를 없앤다.
_focusrect = 0;

// *** 새로운 소켓을 만들고 서버에 접속한다.
function connect () {
// 새로운 XMLSocket 객체를 만든다.
    mySocket = new XMLSocket();

// mySocket의 핸들러에 새로운 콜백 함수를 할당한다.
    mySocket.onConnect = handleConnect;
    mySocket.onClose = handleClose;
    mySocket.onXML = handleIncoming;

// 접속을 시도하고 mySocket.connect()의 리턴 값을 connectSuccess에
// 대입한다(연결의 초기 단계가 완료되면 connect()에서 true를 리턴한다).
    var connectSuccess = mySocket.connect("www.myserver.com", 1025);
    if (connectSuccess) {
        trace("initial connection succeeded");
    } else {
        // connectSuccess가 false, 즉 연결에 실패한 경우
        gotoAndStop("connectionFailed");
        trace("initial connection failed");
    }
}

// *** 연결 시도가 완료되었을 때 실행되는 이벤트 핸들러
function handleConnect (succeeded) {
    // handleConnect()의 succeeded 인자가 true이면 연결에 성공한
```

```

// 것이므로 채팅을 시작하면 된다.
// false이면 연결에 실패했다는 메시지를 출력한다.
if (succeeded) {
    // 연결이 되었음을 나타내는 속성을 true로 설정한다.
    mySocket.connected = true;
    gotoAndStop("chat");
    // 커서를 "send message" 텍스트 필드 위치로 옮긴다.
    Selection.setFocus("_level0.outgoing");
} else {
    // 연결에 실패한 경우에 오류 메시지를 출력한다.
    gotoAndStop("connectionFailed");
    trace("connection failed");
}
}

// *** 서버에서 연결을 끊었을 때 실행되는 이벤트 핸들러
function handleClose () {
    // 사용자에게 연결이 종료되었음을 알린다.
    incoming += ("The server has terminated the connection.\n");
    // 채팅 텍스트 필드를 업데이트했으므로 스크롤 관리자에게 그 사실을 알린다.
    incomingUpdated = true;
    // 연결 상태를 나타내는 속성을 설정한다.
    mySocket.connected = false;
    numClients = 0;
}

// *** 서버에서 메시지를 전달받아 화면에 표시하는 이벤트 핸들러
function handleIncoming (messageObj) {
    // XML 데이터를 Output 창에 표시한다.
    trace("-----new data received-----");
    trace(">>" + messageObj.toString() + "<<");
    trace("----- end of new data -----");

    // 채팅 텍스트 필드를 업데이트했으므로 스크롤 관리자에게 그 사실을 알린다.
    incomingUpdated = true;
    lastScrollPos = incoming.scroll;

    // 시각을 체크한다.
    var now = new Date();
    var hours = now.getHours();
    var minutes = now.getMinutes();
    var seconds = now.getSeconds();

```

```
// 시각을 출력하기 위한 형식에 맞춘다.
hours = (hours < 10 ? "0" : "") + hours;
minutes = (minutes < 10 ? "0" : "") + minutes;
seconds = (seconds < 10 ? "0" : "") + seconds;

// 클라이언트와 접속이 연결될 때와 끊길 때 서버에서 NUMCLIENTS를 보낸다.
// XML 객체에 NUMCLIENTS가 있으면...
if (messageObj.firstChild.nodeName == "NUMCLIENTS") {
// ...incoming 메시지가 창이 비어있는지 확인한다. 그러면...
    if (incoming == "") {
        // ... 사용자가 막 접속한 것이므로 채팅창에 환영 메시지를 출력한다.
        incoming += ("welcome to mock comm 1.0.0, "
            + userID + "\n" + " connection time: " + hours + ":"
            + minutes + ":" + seconds + "\n" + " server: clayton\'s javaComm
            generic flash xmlsocket server\n\n");
    } else {
        // 그렇지 않으면 다른 사용자가 새로 들어왔거나 방을 나간 것이다.
        if (parseInt(messageObj.firstChild.firstChild.nodeValue) >
            numClients) {
            // 채팅창에 새로운 클라이언트가 들어왔음을 알린다.
            incoming += (hours + ":" + minutes + ":"
                + seconds + " a new user has connected.\n");
        } else {
            // 채팅창에 클라이언트가 방을 나갔음을 알린다.
            incoming += (hours + ":" + minutes + ":"
                + seconds + " a user disconnected.\n");
        }
    }
}
// 클라이언트의 수를 다시 세고 환영/환송 사운드를 재생한다.
numClients = parseInt(messageObj.firstChild.firstChild.nodeValue);
welcomeSound.setVolume(100);
welcomeSound.start();
} else {
    // NUMCLIENTS가 없으므로 일반적인 메시지이다.
    // XML 객체에서 사용자 이름과 메시지를 찾아낸다.
    var user = messageObj.firstChild.firstChild.nodeValue;
    var message = messageObj.childNodes[1].firstChild.nodeValue;

    // 메시지를 타임 스탬프와 함께 채팅 창에 추가한다.
    incoming += (hours + ":" + minutes
        + ":" + seconds + user + ">> " + message + "\n");
}
```

```

// 새로운 메시지 클릭 소리를 처리한다.
// 마지막 메시지를 받은지 30초가 지나면 크게 클릭 소리를 낸다.
// 그렇지 않으면 작은 클릭 소리를 낸다.
trace("time since last message: " + (now - lastIncomingMessageTime));
if (lastIncomingMessageTime && (now - lastIncomingMessageTime) >
    30000) {
    click.setVolume(200);
} else {
    click.setVolume(30);
}
click.start();
}

// 메인 채팅 텍스트 필드가 5000 글자를 넘어가면
// 앞 부분을 잘라낸다.
if (incoming.length > 5000) {
    var nearestNewline = incoming.indexOf("\n", incoming.length -
        5000);
    incoming = incoming.substring(nearestNewline, incoming.length);
}

// 다음 번에 사용하기 위해 이 메시지가 도착한 시각을 기록한다.
lastIncomingMessageTime = now;
}

// *** 새로운 XML 객체를 서버로 보내는 함수
function sendMessage() {
    // XML 소스로 보낼 메시지를 만든다.
    // MESSAGE와 USER에 텍스트 자식 노드를 집어넣으려면
    // <USER>와 </MESSAGE> 앞에 공백이 들어가야 한다는 점에 주의하자.
    var message = '<USER> ' + userID + '</USER><MESSAGE>'
        + outgoing + ' </MESSAGE>';

    // 메시지를 XML 객체 계층으로 변환한다.
    messageObj = new XML();
    messageObj.parseXML(message);

    // 보내고 있는 것을 체크한다.
    trace("Sending: " + messageObj);

    // 소켓 객체가 생성되고 접속이 성공적으로 이루어졌다면 XML
    // 객체를 보낸다. 그렇지 않으면 접속해야 한다는 메시지를 출력한다.

```

```
if (mySocket && mySocket.connected) {
    mySocket.send(messageObj);
    // 메시지를 입력하는 텍스트 필드의 내용을 지운다.
    outgoing = "";
} else {
    // 서버에서 연결을 끊은 경우
    incoming += "You are no longer connected. Please reconnect.\n"
    incomingUpdated = true;
}
}

// *** 서버로의 연결을 종료한다.
function quit() {
    if (mySocket.connected) {
        mySocket.close();
        mySocket.connected = false;
        numClients = 0;
        incoming = "";
        gotoAndStop("login");
    }
}
```

참조

loadVariables(), XML 클래스

XMLSocket.close() 메소드

서버 애플리케이션으로 연결된 접속을 종료하는 메소드

버전 플래시 5

문법 socket.close()

설명

close()는 socket과 서버 애플리케이션 사이의 통신 연결을 끊는 메소드이다. socket의 close() 메소드를 실행하고 나면 socket 객체의 send() 메소드를 호출할 수 없다. 또한 서버 애플리케이션에서도 socket을 통해 플래시에 데이터를 보낼 수 없다.

socket 접속이 이미 끊겨 있거나 연결한 적이 없다면, `close()` 메소드를 사용해도 아무런 일이 일어나지 않는다. `close()` 메소드를 호출할 때는 소켓 객체의 `onClose()` 핸들러가 실행되지 않는다(`onClose()` 핸들러는 서버측에서 연결을 끊었을 때만 실행된다).

참조

`XMLSocket.connect()`, `XMLSocket.onClose()`

XMLSocket.connect() 메소드

서버 애플리케이션에 연결하는 메소드

버전	플래시 5
문법	<code>socket.connect(host, port)</code>
인자	
<i>host</i>	“www.myserver.com”과 같은 호스트 이름이나 표준 IP 주소(111.222.3.123과 같이 네 개의 점으로 구분된 8비트 10진수)를 나타내는 문자열. null이나 비어있는 문자열을 <code>host</code> 인자로 사용하면 무비가 제공된 서버의 주소가 기본값으로 쓰인다.
<i>port</i>	1024 이상의 TCP 포트 번호를 나타내는 정수

리턴 값

연결 시도 결과를 나타내는 부울 값. 성공한 경우에는 `true`, 실패한 경우에는 `false`가 리턴된다.

설명

`connect()`는 `host`의 `port`에서 돌아가는 서버 애플리케이션과 플래시 사이의 접속을 시도하는 메소드이다.

`connect()`의 리턴 값이 `true`이면 연결의 초기 단계가 성공적으로 처리되었음을 의미하며 `onConnect()` 핸들러가 호출된다. `onConnect()` 핸들러에서는 연결이 완벽하게 이루어졌는지 확인할 수 있다. 연결을 시도하다 보면 생각보다 시간이 많이 걸릴 수도 있다는 점에 주의하자. `connect()`를 호출할 때는 사용자에게 연결 중이라는 것을 알려주는 것이 좋다.

connect()에서 false가 리턴되면 연결의 초기 단계가 제대로 이루어지지 않았다는 것을 알 수 있다. 이 경우에는 socket의 onConnect() 핸들러가 실행되지 않는다.

연결이 제대로 되었는지 확인하려면, connect() 메소드의 리턴 값을 체크하고 그 값이 true이더라도 onConnect() 핸들러의 success 매개변수까지 확인해야 한다는 점에 주의하자.

주의 사항

connect() 메소드를 이용할 때 보안 문제 때문에 인터넷의 모든 호스트와 연결할 수 없다. 보통 그 무비를 다운로드한 도메인에 있는 호스트에만 접속할 수 있다. connect() 메소드에 적용되는 규칙은 loadVariables() 함수에 적용되는 규칙과 같다. connect() 메소드에 적용되는 도메인 요건의 목록은 loadVariables() 전역 함수 항목에 나온 [표 R-8]에 나와 있다. 보안 관련 제약 조건에 어긋나는 접속을 시도하면 connect() 메소드에서 false를 리턴한다. 독립형 플레이어에서는 보안 관련 제약 조건이 적용되지 않는다.

예제

```
// 새로운 소켓 객체를 만든다.
mySocket = new XMLSocket();
// onConnect에 콜백 핸들러 함수를 대입한다.
mySocket.onConnect = handleConnect;
// myserver.com의 10000번 포트에서 실행되는 애플리케이션에 접속한다.
if (mySocket.connect("myserver.com", 10000) == false) {
    // 오류 메시지가 나오는 프레임으로 이동한다.
    gotoAndStop("failureDialog");
} else {
    // onConnect가 실행될 때까지 기다리기 위한 프레임으로 이동한다.
    gotoAndPlay("connecting");
}
```

참조

XMLSocket.close(), XMLSocket.onConnect()

XMLSocket.onClose() 이벤트 핸들러

서버에서 연결을 끊을 때 실행될 콜백 핸들러를 지정하기 위한 속성

버전 플래시 5

문법
socket.onClose = closeHandler
socket.closeHandler()

설명

onClose 속성을 이용하면 socket의 연결을 서버에서 끊을 때 자동으로 실행시킬 콜백 핸들러를 지정할 수 있다. 서버에서 연결을 끊는 경우는 보통 서버 애플리케이션을 종료시키거나 클라이언트를 고의로 추방하는 경우에 해당한다.

예제

onClose 이벤트에 응답하려면 XMLSocket 객체의 onClose 속성에 사용자가 만든 함수(즉 콜백 핸들러)를 대입해야 한다. 이 콜백 핸들러는 보통 외부에서 소켓 연결을 끊는 것을 알아내기 위한 용도로 쓰인다. 다음 코드에서는 handleClose()를 mySocket의 onClose 속성에 대입한다. 아래 handleClose() 함수는 status 텍스트 필드 값을 업데이트하여 사용자에게 접속이 끊겼음을 알리는 함수이다.

```
mySocket = new XMLSocket();
mySocket.onClose = handleClose;

function handleClose () {
    status += ("\nThe server has terminated the connection.\n");
}
```

참조

XMLSocket.close(); 10장의 '다른 객체에 이벤트 핸들러 추가하기'

XMLSocket.onConnect() 이벤트 핸들러

연결 시도가 완료되었을 때 호출
이벤트 핸들러를 정의하기 위한 속성

버전 플래시 5

문법 `socket.onConnect = connectHandler`
`socket.connectHandler(success)`

인자

success 연결 시도가 성공(true)했는지 실패(false)했는지를 나타내는 부울 값

설명

onConnect는 connect() 메소드 작업이 완료되었을 때 자동으로 실행되는 콜백 핸들러를 지정하기 위한 속성이다. onConnect로 지정한 콜백 핸들러가 실행된다고 해서 연결 작업이 꼭 성공했다고 볼 수는 없다. 연결의 성공 또는 실패와는 상관없이 연결 시도가 완료되지만 하면 콜백 핸들러가 실행되기 때문이다. onConnect로 지정한 콜백 핸들러에는 연결 시도가 성공했는지 여부를 나타내는 success 인자가 전달된다. 제대로 연결된 경우에는 success 값이 true가 된다. 만약 접속에 실패했다면(연결 시간이 경과하거나 서버에서 연결을 거부하는 것과 같은 이유로 연결이 안 되는 경우), success 값이 false가 된다. 액션스크립트에서는 연결이 되지 않는 이유(예: 네트워크 타임아웃, 알 수 없는 호스트, 호스트에서 접속을 거부하는 경우, 기타 연결 오류 등)를 알 수 없다는 점에 주의하자. 따라서 콜백 핸들러는 connect() 명령을 실행하고 나서 적어도 1분 정도 후에(연결하고자 하는 서버의 설정 사항이나 네트워크 속도, 네트워크 트래픽 등에 따라 달라진다) 호출하는 것이 좋다.

예제

onConnect에서 지정한 콜백 핸들러를 이용하여 연결 시도의 성공 여부를 알아낼 수 있다. 실전에서는 연결이 성공된 경우에 데이터 전송을 시작해도 된다는 것을 나타내는 플래그를 설정하기 위한 용도로 콜백 핸들러를 많이 사용한다. 또한 콜백 핸들러를 이용하여 사용자에게 어떤 문제가 있다는 것을 알려주는 것과 같이 오류를 처리하는 코드를 실행시킬 수도 있다.

onConnect 이벤트에 응답하기 위해서는 XMLSocket 객체의 onConnect 속성에 사용자가 만든 함수(즉 콜백 핸들러)를 대입해야 한다. 다음 코드에서는 mySocket

의 `onConnect` 속성에 `handleConnect()` 함수를 대입한다. `handleCode()` 함수에서는 `status` 텍스트 필드의 내용을 업데이트하여 사용자에게 연결 성공 여부를 알려준다.

```
mySocket = new XMLSocket();
mySocket.onConnect = handleConnect;

function handleConnect (succeeded) {
    if (succeeded) {
        status += ("Successfully connected.\n");
    } else {
        status += ("Connection attempt failed.\n");
    }
}
```

더 복잡한 프로그램에서 `onConnect()` 핸들러를 사용하는 방법은 `XMLSocket` 클래스의 예제에 나와 있다.

참조

`XMLSocket.connect()`; 10장의 '다른 객체에 이벤트 핸들러 추가하기'

XMLSocket.onData() 이벤트 핸들러

외부 데이터를 받고 나서 XML로
파싱하기 전에 자동으로 실행되는 메소드

버전 플래시 5

문법 `socket.onData(src)`

인자

src 로딩된 데이터를 포함하고 있는 문자열. 보통 XML 소스 코드를 인자로 사용한다.

설명

`onData()` 핸들러는 `socket`을 통해 플래시에 제로 바이트(ASCII의 널 문자)가 전달될 때마다 자동으로 실행된다. `onData()`는 `src`로부터 새로운 XML 객체 계층을 구성하고 그 계층을 `socket.onXML()`로 보내는 역할을 한다. 하지만 `onData()` 핸

들러에 사용자가 직접 만든 콜백 함수를 대입하여 액션스크립트에서 주어진 문자열을 XML로 파싱하기 전에 다른 작업을 처리할 수도 있다. 상황에 따라(예를 들면 실시간 비디오 게임같은 경우) 액션스크립트에 내장된 XML 파싱 기능을 이용하는 것보다 사용자가 직접 src의 데이터를 처리하는 것이 더 빠를 수도 있다.

예제

다음 코드에는 onData()에 사용자 정의 콜백 함수를 대입하는 방법이 나와 있다. 아래에서 정의한 콜백 함수는 mySocket에서 받은 데이터를 화면에 출력하고 액션스크립트에서 XML로 파싱을 하지 못하도록 하는 매우 단순한 함수이다.

```
mySocket = new XMLSocket();

mySocket.onData = function (src) {
    trace("Received data: \n" + src);
};
```

참조

XMLSocket.onXML()

XMLSocket.onXML() 이벤트 핸들러

XMLSocket 객체에서 데이터를 받고 XML로 파싱한 다음에 호출할 콜백 핸들러를 정의하기 위한 속성

버전 플래시 5

문법 socket.onXML = xmlHandler
socket.xmlHandler(XMLObject)

인자

XMLObject 외부에서 들어온 XML 형식의 데이터를 저장할 XML 객체

설명

onXML은 플래시에서 외부로부터 XML을 받았을 때 실행시킬 콜백 핸들러를 지정하기 위한 속성이다. socket에서 서버로부터 데이터 블록(즉 문자열 뒤에 ASCII 널 문자가 붙어있는 것)을 전달받으면, socket.onXML로 지정한 콜백 핸들러가 자동 호

출된다. 서버에서는 데이터를 아무 때나 보낼 수 있지만, 콜백 핸들러는 socket에서 받은 데이터의 맨 뒤에 널 문자(제로 바이트)를 받았을 때만 실행된다. 자바에서는 제로 바이트를 '\0'으로 표기한다. 제로 바이트를 받으면 액션스크립트에서 그 전 제로 바이트 이후에 받은 데이터(이번에 받은 것이 첫 번째 제로 바이트라면 연결된 후에 받은 모든 데이터)를 파싱한다. 이렇게 파싱한 데이터는 XML 객체 계층으로 변환되고 이 계층은 다시 콜백 핸들러의 XMLObject 인자로 전달된 객체에 저장된다.

클라이언트/서버 애플리케이션에서 클라이언트측만 말고 있는 플래시 프로그램이라면 onXML로 지정한 콜백 핸들러에서는 새로 도착한 XML 데이터를 받아서 처리한다는 점만 기억해두면 된다. 새로 받은 XML 데이터는 XMLObject를 통해 액세스할 수 있다.

소켓을 통해 전송된 데이터를 파싱하지 않고 그대로 사용하고 싶다면, 소켓의 onData() 핸들러를 새로 만들어서 사용하면 된다(XMLSocket.onData() 참조).

예제

onXML 이벤트에 응답하려면 XMLSocket 객체의 onXML 속성에 사용자가 만든 함수(콜백 핸들러)를 대입해야 한다. 다음 코드에서는 mySocket의 onXML 속성에 handleIncoming() 함수를 대입한다. handleIncoming() 함수에서는 messageObj에 저장된 XML 객체 계층 중 한 노드를 액세스하여 그 값을 messages라는 텍스트 필드에 추가한다.

```
mySocket = new XMLSocket();
mySocket.onXML = handleIncoming;

function handleIncoming (messageObj) {
    trace("Got some new data!");
    // messageObj에 <MESSAGE>text</MESSAGE> 라는 XML 코드가 저장된다.
    var message = messageObj.firstChild.firstChild;
    messages += (message.nodeValue + "\n");
}
```

조금 더 복잡한 프로그램에서 onXML 코드를 사용하는 법은 XMLSocket 클래스 예제에 수록되어 있다.

참조

XMLSocket.send, XMLSocket.onData(); 10장의 '다른 객체에 이벤트 핸들러 추가하기'

XMLSocket.send() 메소드

XML 형식의 데이터를 서버 애플리케이션으로 전송하는 메소드

버전 플래시 5

문법 socket.send(XMLObject)

인자

XMLObject 문자열로 변환하여 서버 애플리케이션으로 전송할 XML 객체 또는 XML 형식의 텍스트를 포함하고 있는 문자열

설명

send()는 socket을 통해 플래시에서 서버 애플리케이션으로 메시지를 전송하는 메소드이다. 서버로 보낼 메시지는 XML 클래스의 객체여야 하지만 문자열을 대신 사용해도 된다. send()를 호출하면 XMLObject는 문자열로 변환되어 원격 애플리케이션으로 전송되며, 이 때 문자열의 맨 뒤에 제로 바이트(ASCII 널 문자)가 추가된다. 원격 애플리케이션에서 반드시 응답을 보내야 하는 것은 아니다. 애플리케이션에서 응답을 보내오면 socket의 onXML() 이벤트 핸들러가 실행된다.

예제

아래 코드는 mySocket이라는 소켓(서버에 접속되어 있는 XMLSocket 객체)을 통해 원격 객체에 간단한 XML 형식의 메시지를 전송하는 예이다. 이 때 message는 XMLSocket이 아닌 XML 클래스에 속하는 객체라는 점에 주의하자(XMLSocket과 관련된 예제는 XMLSocket 클래스 항목에서 찾을 수 있다).

```
var message = new XML('<MESSAGE>testing...testing...</MESSAGE>');
mySocket.send(message);
```

XML 객체로 감싸지 않은 채 XML 형식의 텍스트를 포함하는 문자열을 그대로 전송할 수도 있다. 간단한 XML 메시지인 경우에는 다음과 같은 식으로 직접 보내도 된다.

```
mySocket.send('<MESSAGE>testing...testing...</MESSAGE>');
```

참조

XMLSocket.onXML; XMLSocket 클래스, XML.send()