ACTIONSCRIPT™ 3.0 프로그래밍

Adobe

ActionScript™ 3.0 프로그래밍

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

목 차

설명서 정보

이 설명서는 ActionScript™ 3.0을 사용하여 응용 프로그램을 개발하는 데 필요한 기본 정보를 제공합니다. 여기에 설명된 개념과 기술을 이해하기 위해서는 데이터 유형, 변수, 루프, 함수 등의 일반 프로그래밍 개념에 대한 지식이 필요하며, 기본적인 객체 지향 프로그래밍 개념(예: 클래스, 상속 등)에 대해 이해해야 합니다. ActionScript 1.0 또는 ActionScript 2.0에 대한 사전 지식이 있으면 도움이 되지만 반드시 필요한 것은 아닙니다.

목차

설명서 사용

이 설명서의 장은 ActionScript 설명서의 관련 영역을 쉽게 찾을 수 있도록 다음과 같은 논리 그룹으로 구성되어 있습니다.

장	설명
1-4장: ActionScript 프로그래밍 개요	언어 구문, 문, 연산자 등의 핵심 ActionScript 3.0 개념, ECMAScript 버전 4 초안 언어 사양, 객체 지향 ActionScript 프로그래밍, Adobe® Flash® Player 9 표시 목록의 표시 객체 관리에 대한 새로운 접근 방식에 대해 설명합니다.
5-10장: 기본 ActionScript 3.0 데이터 유형 및 클래스	ECMAScript 초안 사양의 일부이기도 한 ActionScript 3.0의 최상위 데이터 유형에 대해 설명합니다.
11-26장: Flash Player API	이벤트 처리, 네트워킹 및 통신, 파일 입/출력, 외부 인터페이스, 응용 프로그램 보안 모델 등 Adobe Flash Player 9의 고유 패키지 및 클래스에서 구현된 주요 기능에 대해 설명합니다.

이 설명서에는 중요 클래스 및 일반적으로 사용되는 클래스에 대한 응용 프로그램 프로그래밍 개념을 보여 주는 여러 샘플 파일이 포함되어 있습니다. 샘플 파일은 로드하여 Adobe® Flash® CS3 Professional과 함께 사용하기 쉽도록 패키지화되어 있으며 래퍼 파일이 들어 있을 수 있습니다. 하지만 기본 샘플 코드는 사용자가 선호하는 모든 개발 환경에서 사용할 수 있는 순수 ActionScript 3.0입니다.

다음과 같은 여러 가지 방법으로 ActionScript 3.0을 작성하고 컴파일할 수 있습니다.

- Adobe Flex Builder 2 개발 환경 사용
- Flex Builder 2와 함께 제공된 명령줄 컴파일러 및 텍스트 편집기 사용
- Adobe® Flash® CS3 Professional 제작 도구 사용

ActionScript 개발 환경에 대한 자세한 내용은 제1장, “ActionScript 3.0 소개”를 참조하십시오.

본 설명서의 코드 샘플을 이해하기 위해 Flex Builder 또는 Flash 제작 도구와 같은 ActionScript의 통합 개발 환경 사용에 대한 경험이 필요한 것은 아닙니다. 그러나 관련 설명서에서 이러한 개발 환경을 사용하여 ActionScript 3.0 코드를 작성하고 컴파일하는 방법을 참조할 수 있습니다. 자세한 내용은 15페이지의 “ActionScript 설명서 액세스”를 참조하십시오.

ActionScript 설명서 액세스

이 설명서는 강력한 기능을 갖춘 객체 지향 프로그래밍 언어인 ActionScript 3.0을 설명하는 데 초점을 두고 있으므로, 특정 도구나 서버 아키텍처 내의 응용 프로그램 개발 프로세스 또는 작업 과정에 대해서는 상세하게 다루지 않습니다. 따라서 ActionScript 3.0 응용 프로그램 설계, 개발, 테스트 및 배포를 수행하는 동안 *ActionScript 3.0 프로그래밍* 외에 다른 설명서를 참조할 수 있습니다.

ActionScript 3.0 설명서

이 설명서를 참조하면 ActionScript 3.0 프로그래밍 언어와 관련된 개념을 이해할 수 있고 상세한 구현 정보 및 중요 언어 기능 샘플도 활용할 수 있습니다. 하지만 이 설명서는 전체 언어 참조 설명서가 아닙니다. 전체 언어 참조를 보려면 이 언어로 된 모든 클래스, 메서드, 속성, 이벤트 등을 설명하는 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오. *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서는 기본 언어, Flash 구성 요소(fl 패키지), Flash Player API(flash 패키지) 등에 대한 자세한 참조 정보가 제공됩니다.

Flash 설명서

Flash 개발 환경을 사용하는 경우 다음 설명서를 참조할 수 있습니다.

설명서	설명
Flash 사용 설명서	Flash 제작 환경에서 동적인 웹 응용 프로그램을 개발하는 방법을 설명합니다.
ActionScript 3.0 프로그래밍	ActionScript 3.0 언어 및 기본 Flash Player API의 특정 사용 방법을 설명합니다.
ActionScript 3.0 언어 및 구성 요소 참조 설명서	Flash 구성 요소 및 ActionScript 3.0 API의 구문, 사용 방법, 코드 예제 등을 제공합니다.
ActionScript 3.0 구성 요소 사용 설명서	구성 요소를 사용하여 Flash 응용 프로그램을 개발하는 방법에 대해 자세히 설명합니다.
Adobe Flash에서 ActionScript 2.0 학습	ActionScript 2.0 구문 개요를 제공하며 서로 다른 객체 유형 작업 시 ActionScript 2.0를 사용하는 방법을 설명합니다.
ActionScript 2.0 언어 참조 설명서	Flash 구성 요소 및 ActionScript 2.0 API의 구문, 사용 방법, 코드 예제 등을 제공합니다.
ActionScript 2.0 구성 요소 사용 설명서	ActionScript 2.0 구성 요소를 사용하여 Flash 응용 프로그램을 개발하는 방법에 대해 자세히 설명합니다.

설명서	설명
ActionScript 2.0 구성 요소 언어 참조 설명서	버전 2 Adobe Component Architecture에서 사용할 수 있는 각 구성 요소 및 API를 설명합니다.
Flash 확장	JavaScript API에서 사용할 수 있는 객체, 메서드 및 속성에 대해 설명합니다.
Flash Lite 2.x 시작	Adobe® Flash® Lite™ 2.x를 사용하여 응용 프로그램을 개발하는 방법을 설명하고, Flash Lite 2.x에서 사용할 수 있는 ActionScript 기능의 구문, 사용 방법, 코드 예제 등을 설명합니다.
Flash Lite 2.x 응용 프로그램 개발	Flash Lite 2.x 응용 프로그램을 개발하는 방법을 설명합니다.
Flash Lite 2.x ActionScript 소개	Flash Lite 2.x를 사용하여 응용 프로그램을 개발하는 방법을 소개하고, Flash Lite 2.x 개발자가 사용할 수 있는 ActionScript 기능을 모두 설명합니다.
Flash Lite 2.x ActionScript 언어 참조 설명서	Flash Lite 2.x에서 사용할 수 있는 ActionScript 2.0의 구문, 사용 방법, 코드 예제 등을 제공합니다.
Flash Lite 1.x 시작	Flash Lite 1.x를 소개하고 Adobe® Device Central CS3 에뮬레이터를 사용하여 내용을 테스트하는 방법을 설명합니다.
Flash Lite 1.x 응용 프로그램 개발	Flash Lite 1.x를 사용하여 휴대 장치용 응용 프로그램을 개발하는 방법을 설명합니다.
Flash Lite 1.x ActionScript 학습	Flash Lite 1.x 응용 프로그램에서 ActionScript를 사용하는 방법뿐 아니라 Flash Lite 1.x에서 사용할 수 있는 ActionScript 기능을 모두 설명합니다.
Flash Lite 1.x ActionScript 언어 참조 설명서	Flash Lite 1.x에서 사용할 수 있는 ActionScript 요소의 구문과 사용 방법을 제공합니다.

ActionScript 학습 리소스

Adobe에서는 이 설명서의 내용 이외에도 Adobe 개발자 센터 및 Adobe 디자인 센터를 통해 정기적으로 업데이트되는 문서, 디자인 아이디어 및 예제를 제공합니다.

Adobe 개발자 센터

Adobe 개발자 센터는 ActionScript에 대한 최신 정보, 실제 응용 프로그램 개발 관련 기사 및 중요하게 부각되고 있는 이슈에 대한 정보를 제공하는 리소스입니다. 개발자 센터는 www.adobe.com/devnet_kr에 있습니다.

Adobe 디자인 센터

최신 디지털 디자인 및 모션 그래픽을 만나보십시오. 주요 아티스트별로 작품을 찾아보고 새로운 디자인 흐름을 살펴보며 자습서, 핵심 작업 과정 및 고급 기술을 통해 기술 향상을 도모할 수 있습니다. 한 달에 두 번씩, 새로운 자습서와 기사 및 창조적 영감을 불러일으키는 갤러리 작품들이 전시됩니다. 디자인 센터는 www.adobe.com/designcenter_kr에 있습니다.

제 1 장

ActionScript 3.0 소개

1

이 장에서는 혁신적인 기능이 포함된 ActionScript의 최신 버전인 ActionScript 3.0에 대한 개요를 설명합니다.

목차

ActionScript

ActionScript는 Adobe Flash Player 런타임 환경의 프로그래밍 언어입니다. 이를 사용하여 Flash 내용과 응용 프로그램에서 대화형 작업, 데이터 처리 등 보다 다양한 작업을 수행할 수 있습니다.

ActionScript는 Flash Player에 내장되어 있는 AVM(ActionScript Virtual Machine)에 의해 실행됩니다. ActionScript 코드는 일반적으로 Adobe Flash CS3 Professional 또는 Adobe® Flex™ Builder™에 내장되어 있는 컴파일러나 Adobe® Flex™ SDK 및 Flex™ Data Services에서 제공되는 컴파일러에 의해 *바이트코드 형식*(컴퓨터가 기록하고 이해하는 프로그래밍 언어의 일종)으로 컴파일됩니다. 바이트코드는 SWF 파일에 포함되며 Flash Player 런타임 환경에서 실행됩니다.

ActionScript 3.0에서는 객체 지향 프로그래밍에 대한 기본적인 지식만을 가진 개발자도 손쉽게 사용할 수 있는 강력한 프로그래밍 모델을 제공합니다. ActionScript 3.0의 주요 기능을 몇 가지 소개하면 다음과 같습니다.

- 새 바이트코드 명령어 집합을 사용하고 성능이 크게 향상된 새로운 AVM(ActionScript Virtual Machine)인 AVM2
- 보다 철저하게 ECMAScript(ECMA 262) 표준을 따르고 이전 버전의 컴파일러보다 심도 있는 최적화를 수행하는 최신 컴파일러 코드 베이스

- 객체에 대한 저수준 제어 및 진정한 객체 지향 모델을 사용하여 확장되고 향상된 API(Application Programming Interface)
- 다음 ECMAScript(ECMA-262) Edition 4 초안 언어 사양을 기반으로 하는 기본 언어
- E4X(ECMAScript for XML) 사양(ECMA-357 edition 2)을 기반으로 한 XML API. E4X는 언어의 기본 데이터 유형으로 XML이 추가된 ECMAScript의 언어 확장입니다.
- DOM(Document Object Model) 레벨 3 이벤트 사양을 기반으로 하는 이벤트 모델

ActionScript 3.0의 장점

ActionScript 3.0은 이전 버전의 ActionScript를 능가하는 스크립팅 기능을 가지고 있습니다. 대용량 데이터 세트 및 재사용 가능한 객체 지향 코드 베이스를 통해 매우 복잡한 응용 프로그램을 쉽게 만들 수 있습니다. Adobe Flash Player 9에서 내용을 실행하는 데 ActionScript 3.0이 반드시 필요한 것은 아니지만 새로운 가상 머신인 AVM2의 향상된 기능을 활용하려면 ActionScript 3.0을 사용해야 합니다. ActionScript 3.0 코드는 이전 ActionScript 코드보다 실행 속도가 최고 10배 향상되었습니다.

이전 버전의 AVM 즉, AVM1에서는 ActionScript 1.0 및 ActionScript 2.0 코드를 실행합니다. 이전 버전에서 작성한 기존 내용 및 이전 내용과의 호환성을 위해 Flash Player 9에서 AVM1을 지원합니다. 자세한 내용은 24페이지의 “이전 버전과의 호환성”을 참조하십시오.

ActionScript 3.0의 새로운 기능

ActionScript 3.0에 ActionScript 프로그래머에게 익숙한 여러 가지 클래스와 기능이 포함되어 있지만 ActionScript 3.0은 이전 버전의 ActionScript와 구조적 및 개념적인 면에서 다릅니다. ActionScript 3.0의 향상된 기능에는 객체에 대한 저수준 제어를 강화한 Flash Player API 및 기본 언어의 새로운 기능이 포함됩니다.

기본 언어 기능

기본 언어에는 명령문, 표현식, 조건문, 반복문 및 유형과 같은 프로그래밍 언어의 기본적인 구성 단위가 정의되어 있습니다. ActionScript 3.0에는 개발 작업의 속도를 향상시킬 수 있는 여러 가지 새로운 기능이 포함되어 있습니다.

런타임 예외

ActionScript 3.0에서는 이전 버전의 ActionScript보다 많은 오류 상황을 보고합니다. 일반적인 오류 상황에 런타임 예외가 사용되면서 디버깅이 편리해지고 오류를 철저히 처리하는 응용 프로그램을 개발할 수 있게 되었습니다. 런타임 오류를 통해 소스 파일 및 줄 번호 정보가 포함된 스택 추적을 확인하여 오류를 빠르고 정확하게 찾아낼 수 있습니다.

런타임 유형

ActionScript 2.0에서는 모든 값의 유형이 런타임에 동적으로 지정되었기 때문에 유형 약어는 주로 개발자가 참조하는 데 쓰였습니다. 반면 ActionScript 3.0에서는 런타임에 유형 정보가 보존되어 여러 가지 용도로 활용됩니다. Flash Player 9는 런타임에 유형을 확인하여 시스템의 유형 안전을 향상시킵니다. 또한 유형 정보는 변수를 기본 시스템 표현으로 나타내는 데 사용되므로 성능을 향상시키고 메모리 사용량을 줄여 줍니다.

봉인 클래스

ActionScript 3.0에 봉인 클래스 개념이 도입되었습니다. 봉인 클래스에는 컴파일 타임에 정의된 속성 및 메서드의 고정된 집합만 있으며 속성 및 메서드를 더 이상 추가할 수 없습니다. 이로 인해 컴파일 타임에 보다 엄격한 확인 절차가 수행되어 더욱 견고한 프로그램을 만들 수 있게 됩니다. 또한 각 객체 인스턴스에 내부 해시 테이블이 필요하지 않으므로 사용 가능한 메모리가 증가합니다. `dynamic` 키워드를 사용하여 동적 클래스를 정의할 수도 있습니다. ActionScript 3.0의 모든 클래스는 기본적으로 봉인되지만 `dynamic` 키워드를 사용하여 동적으로 선언할 수 있습니다.

메서드 클로저

ActionScript 3.0에서는 메서드 클로저를 원본 객체 인스턴스에 바인딩할 수 있습니다. 이 기능은 이벤트 처리에 유용합니다. ActionScript 2.0에서는 메서드 클로저가 어떠한 객체 인스턴스에서 추출되었는지 기억하지 못했기 때문에 메서드 클로저가 호출되면 예기치 못한 비헤이비어가 발생했습니다. 이 경우 일반적으로 mx.utils.Delegate 클래스를 사용하여 문제를 해결했지만 이제 더 이상 mx.utils.Delegate 클래스를 사용할 필요가 없습니다.

E4X(ECMAScript for XML)

ActionScript 3.0은 최근 ECMA-357로 표준화된 E4X(ECMAScript for XML)를 구현합니다. E4X는 XML을 조작하기 위한 자연스럽고 쉬운 언어 구문 집합을 제공합니다. 기존 XML 구문 분석 API와 달리 E4X에서의 XML은 언어의 기본 데이터 유형처럼 동작합니다. E4X는 필요한 코드 양을 현저하게 줄여 XML을 조작하는 응용 프로그램을 효율적으로 개발할 수 있습니다. ActionScript 3.0에서 E4X 구현에 대한 자세한 내용은 323페이지의 제11장, “XML을 사용한 작업”을 참조하십시오.

ECMA의 E4X 사양을 보려면 www.ecma-international.org를 방문하십시오.

일반 표현식

ActionScript 3.0에는 일반 표현식에 대해 기본적인 지원이 포함되어 있어 문자열을 빠르게 검색하고 조작할 수 있습니다. ActionScript 3.0은 ECMAScript(ECMA-262) Edition 3 언어 사양에 정의되어 있는 일반 표현식을 지원합니다.

네임스페이스

네임스페이스는 선언의 가시성을 제어하는 데 사용되는 기존 액세스 지정자(`public`, `private`, `protected`)와 유사합니다. 네임스페이스는 사용자 정의 액세스 지정자로 작동하며 네임스페이스의 이름은 임의로 지정할 수 있습니다. 네임스페이스는 충돌을 방지하기 위해 URI(Universal Resource Identifier)와 함께 사용되며 E4X를 사용하여 작업하는 경우 XML 네임스페이스를 나타내는 데 사용될 수도 있습니다.

새 프리미티브 유형

ActionScript 2.0에는 Number, 배정밀도 숫자, 부동 소수점 숫자 등으로 불리는 단 하나의 숫자 유형이 있습니다. ActionScript 3.0에는 int 및 uint 유형이 추가되었습니다. int 유형은 ActionScript 코드에서 CPU의 신속한 정수 계산 기능을 이용할 수 있는 부호 있는 32비트 정수입니다. int 유형은 정수가 사용되는 루프 카운터 및 변수에 적합합니다. uint 유형은 RGB 색상 값, 바이트 수 등에 적합한 부호 없는 32비트 정수 유형입니다.

Flash Player API 기능

ActionScript 3.0의 Flash Player API에는 저수준에서 객체를 제어할 수 있는 여러 가지 새로운 클래스가 포함되어 있습니다. 언어의 구조는 완전히 새롭고 보다 직관적입니다. 새로운 클래스가 너무 많아 여기에서 자세하게 다룰 수 없지만 다음 단원에서 몇 가지 중요한 변경 사항에 대해 설명합니다.

DOM3 이벤트 모델

DOM3(Document Object Model Level 3) 이벤트 모델에서 이벤트 메시지를 생성하고 처리하는 표준 방식을 제공하므로 응용 프로그램 내에서 객체 간 상호 작용 및 통신이 가능해져 객체의 상태를 유지하고 변경 사항에 대응할 수 있습니다. W3C(World Wide Web Consortium) DOM Level 3 이벤트 사양을 기반으로 만들어진 이 이벤트 모델은 이전 버전의 ActionScript에서 제공되는 이벤트 시스템보다 명확하고 효율적인 메커니즘을 제공합니다.

이벤트 및 오류 이벤트는 flash.events 패키지에 있습니다. Flash 구성 요소 프레임워크에 Flash Player API와 동일한 이벤트 모델이 사용되므로 전체 Flash 플랫폼에서 일관된 이벤트 시스템이 사용됩니다.

표시 목록 API

Flash Player 표시 목록 즉, Flash 응용 프로그램의 시각적인 요소가 모두 포함되어 있는 트리에 액세스하는 데 사용할 수 있는 API는 Flash의 시각적인 프리미티브 값을 처리하는 클래스로 구성되어 있습니다.

새로운 Sprite 클래스는 MovieClip 클래스와 유사하지만 UI 구성 요소의 기본 클래스로서 더욱 적합한 경량급 구성 단위입니다. 새로운 Shape 클래스는 벡터의 모양을 그대로 나타냅니다. 이러한 클래스는 `new` 연산자를 사용하여 무리 없이 인스턴스화될 수 있으며 언제든지 동적으로 다른 클래스의 하위 클래스가 될 수 있습니다.

이제 심도 관리 기능이 Flash Player에 내장되어 자동으로 수행되므로 더 이상 심도 값을 지정할 필요가 없습니다. 객체의 z-순서를 지정하고 관리할 수 있는 새로운 메서드가 제공됩니다.

동적 데이터 및 내용 처리

ActionScript 3.0에는 Flash 응용 프로그램에서 에셋 및 데이터를 로드하고 처리할 수 있는 메커니즘이 포함되어 있습니다. 이 메커니즘은 직관적이며 API 전체에서 일관성을 유지합니다. 새로운 Loader 클래스는 SWF 파일과 이미지 에셋을 로드할 수 있는 단일 메커니즘 및 로드된 내용에 대한 세부 정보에 액세스할 수 있는 방법을 제공합니다. URLLoader 클래스는 데이터 기반 응용 프로그램에서 텍스트와 이진 데이터를 로드할 수 있는 별도의 메커니즘을 제공합니다. Socket 클래스는 포맷에 관계없이 서버 소켓에서 이진 데이터를 읽고 쓸 수 있는 방법을 제공합니다.

저수준 데이터 액세스

다양한 API에서 이전의 ActionScript에서는 사용할 수 없었던 데이터에 대한 저수준 액세스를 제공합니다. URLLoader에서 구현되는 URLStream 클래스는 데이터를 다운로드하는 동안 해당 데이터를 원시 이진 데이터로 액세스할 수 있게 합니다. ByteArray 클래스를 사용하면 이진 데이터에 대한 읽기, 쓰기 및 작업 기능을 최적화할 수 있습니다. 새로운 Sound API의 SoundChannel 및 SoundMixer 클래스를 통해 사운드를 보다 정밀하게 제어할 수 있습니다. 보안을 처리하는 새로운 API에서는 보안 오류를 해결할 수 있도록 SWF 파일 또는 로드된 내용의 보안 권한에 대한 정보를 제공합니다.

텍스트를 사용한 작업

ActionScript 3.0에는 모든 텍스트 관련 API를 묶은 flash.text 패키지가 포함되어 있습니다. ActionScript 2.0의 `TextField.getLineMetrics()` 메서드를 대체하는 TextLineMetrics 클래스는 텍스트 필드 내의 텍스트 행에 대한 자세한 메트릭을 제공합니다. TextField 클래스에는 새롭고 흥미로운 저수준 메서드가 여러 개 포함되어 있습니다. 이러한 메서드는 텍스트 필드에 있는 텍스트 줄 또는 단일 문자에 대한 특정 정보를 제공할 수 있습니다. 이러한 메서드에는 문자의 경계 상자를 나타내는 사각형을 반환하는 `getCharBoundaries()`, 지정된 위치에 있는 문자의 인덱스를 반환하는 `getCharIndexAtPoint()` 및 단락에 있는 첫 번째 문자의 인덱스를 반환하는 `getFirstCharInParagraph()`가 있습니다. 행 수준의 메서드에는 지정된 텍스트 줄의 문자 수를 반환하는 `getLineLength()` 및 지정된 줄의 텍스트를 반환하는 `getLineText()`가 있습니다. 새로운 Font 클래스는 SWF 파일에 포함된 글꼴을 관리하는 방법을 제공합니다.

이전 버전과의 호환성

Flash Player에서는 이전에 게시된 내용과의 호환성이 완벽하게 유지됩니다. 이전 버전의 Flash Player에서 실행된 모든 내용이 Flash Player 9에서 실행됩니다. 그러나 Flash Player 9에 ActionScript 3.0이 도입되면서 Flash Player 9에서 실행되는 이전 내용과 새로운 내용 간의 호환성에 일부 문제점이 나타났습니다. 발생할 수 있는 호환성 문제는 다음과 같습니다.

- 단일 SWF 파일에서는 ActionScript 1.0 또는 2.0 코드를 ActionScript 3.0 코드와 결합할 수 없습니다.
- ActionScript 3.0 코드는 ActionScript 1.0 또는 2.0으로 작성된 SWF 파일을 로드할 수 있지만 SWF 파일의 변수 및 함수에 액세스할 수 없습니다.
- ActionScript 1.0 또는 2.0으로 작성된 SWF 파일에서는 ActionScript 3.0으로 작성된 SWF 파일을 로드할 수 없습니다. 이는 Flash 8 또는 Flex Builder 1.5 이하 버전에서 작성된 SWF 파일에서 ActionScript 3.0 SWF 파일을 로드할 수 없음을 의미합니다.

  이 규칙에 대한 유일한 예외는 특정 ActionScript 2.0 SWF 파일에 의해 로드된 파일이 어떠한 레벨에서도 전혀 없는 경우 해당 ActionScript 2.0 SWF 파일을 ActionScript 3.0 SWF 파일로 대체할 수 있다는 것입니다. ActionScript 2.0 SWF 파일에서는 `loadMovieNum()`의 `level` 매개 변수에 값 0을 전달하여 호출함으로써 이를 수행할 수 있습니다.
- 일반적으로 ActionScript 3.0으로 작성된 SWF 파일과 함께 작동할 ActionScript 1.0 또는 2.0으로 작성된 SWF 파일은 마이그레이션해야 합니다. 예를 들어, ActionScript 2.0을 사용하여 미디어 플레이어를 만들었다고 가정해 봅니다. 미디어 플레이어에서 ActionScript 2.0을 사용하여 만들어진 다양한 내용을 로드합니다. ActionScript 3.0에서 만든 새로운 내용은 미디어 플레이어에서 로드할 수 없습니다. 비디오 플레이어를 ActionScript 3.0으로 마이그레이션해야 합니다.

  그러나 ActionScript 3.0으로 미디어 플레이어를 만드는 경우 해당 미디어 플레이어에서 ActionScript 2.0으로 만든 내용을 간단히 로드할 수 있습니다.

다음 표에는 새로운 내용 및 실행 코드의 로드와 관련된 이전 버전의 Flash Player에 대한 제한 사항 및 다른 버전의 ActionScript로 작성된 SWF 파일 간의 크로스 스크립팅에 대한 제한 사항이 요약되어 있습니다.

지원되는 기능	**런타임 환경**		
	Flash Player 7	Flash Player 8	Flash Player 9
로드 가능한 SWF의 Flash Player 버전	7 이하	8 이하	9 이하
포함된 AVM	AVM1	AVM1	AVM1 및 AVM2
실행할 SWF의 ActionScript 버전	1.0 및 2.0	1.0 및 2.0	1.0, 2.0 및 3.0

지원되는 기능*	**내용 작성 버전**	
	ActionScript 1.0 및 2.0	ActionScript 3.0
로드 및 코드 실행 가능한 내용의 작성 버전	ActionScript 1.0 및 2.0만	ActionScript 1.0, 2.0 및 ActionScript 3.0
크로스 스크립팅 가능한 내용의 작성 버전	ActionScript 1.0 및 2.0만†	ActionScript 3.0‡

* Flash Player 9 이상에서 실행되는 내용입니다. Flash Player 8 이하에서 실행되는 내용은 ActionScript 1.0 및 2.0만 로드, 표시, 실행 및 크로스 스크립팅할 수 있습니다.

† LocalConnection을 사용하는 ActionScript 3.0

‡ LocalConnection을 사용하는 ActionScript 1.0 및 2.0

제 2 장

ActionScript 시작

2

이 장에서는 ActionScript 프로그래밍의 기초를 살펴 보고 이 설명서의 나머지 부분에 있는 개념과 예제를 이해하는 데 필요한 배경 지식을 알려 줍니다. ActionScript에서 어떻게 적용되는지를 중심으로 기본적인 프로그래밍 개념을 살펴보고, ActionScript 응용 프로그램을 구성하는 방법에 대한 필수 사항을 설명합니다.

목차

프로그래밍 기본 사항

ActionScript는 프로그래밍 언어이기 때문에 먼저 몇 가지 일반적인 컴퓨터 프로그래밍 개념을 이해하면 ActionScript를 익히는 데 도움이 됩니다.

컴퓨터 프로그램으로 수행하는 작업

먼저 컴퓨터 프로그램이란 무엇이며 이를 통해 수행할 수 있는 작업은 무엇인지에 대한 개념을 정립하는 것이 좋습니다. 컴퓨터 프로그램은 다음과 같은 두 가지 측면에서 고찰할 수 있습니다.

- 프로그램은 컴퓨터에서 수행할 일련의 명령 또는 단계입니다.
- 각 단계는 궁극적으로 정보 또는 데이터의 일부를 조작하는 것과 관련이 있습니다.

일반적으로 컴퓨터 프로그램은 컴퓨터에 지정하는 단계별 명령 목록이며 이러한 명령은 하나씩 수행됩니다. 각 개별 명령을 *명령문*이라고 합니다. 이 설명서 전체에서 볼 수 있듯이 ActionScript에서는 각 명령문의 끝에 세미콜론을 사용합니다.

사실 프로그램에 지정된 명령에 의해 수행되는 작업은 컴퓨터 메모리에 저장된 데이터의 일부 비트를 조작하는 것이 전부입니다. 간단한 예로 두 숫자를 더하고 그 결과를 메모리에 저장하도록 컴퓨터에게 지시하는 경우를 생각해 볼 수 있습니다. 보다 복잡한 예로 스크린에 그려진 사각형을 스크린의 다른 영역으로 이동하도록 프로그램을 작성하려는 경우를 가정해 봅시다. 컴퓨터에서 사각형의 x 및 y 좌표, 너비와 높이, 색상 등 사각형에 대한 특정 정보를 추적합니다. 이러한 정보의 각 비트가 컴퓨터 메모리에 저장됩니다. 사각형을 다른 위치로 이동하는 프로그램은 "x 좌표를 200으로 변경하고 y 좌표를 150으로 변경하는" 즉, x 및 y 좌표에 사용할 새 값을 지정하는 단계로 구성됩니다. 물론 컴퓨터에서는 해당 숫자를 컴퓨터 화면에 나타나는 이미지로 변환하기 위해 좌표 데이터를 사용하여 어떤 작업을 하겠지만 현 단계에서는 이에 대한 세부 사항을 몰라도 되며 "스크린에서 사각형 이동" 과정을 수행하면 컴퓨터 메모리에 있는 데이터의 비트가 변경된다는 정도만 알고 있으면 충분합니다.

변수 및 상수

프로그래밍은 주로 컴퓨터 메모리의 여러 정보를 변경하는 것과 관련되므로 프로그램에서 단일 정보를 나타내는 방법이 필요합니다. *변수*는 컴퓨터 메모리에 있는 값을 나타내는 이름입니다. 값을 조작하기 위해 명령문을 작성할 때 값 대신 변수 이름을 사용합니다. 이렇게 하면 프로그램 내에서 변수 이름이 식별될 때마다 컴퓨터에서 메모리를 조회하여 발견된 값을 사용합니다. 예를 들어, 숫자가 저장된 `value1`과 `value2`라는 두 개의 변수가 있는 경우 해당 숫자를 추가하기 위해 다음과 같은 명령문을 작성할 수 있습니다.

```
value1 + value2
```

실제로 일련의 단계가 실행되면 컴퓨터에서 각 변수의 값을 확인하고 두 값을 더합니다.

ActionScript 3.0에서 실제로 변수는 다음과 같은 세 부분으로 구성됩니다.

- 변수 이름
- 변수에 저장할 수 있는 데이터의 유형
- 컴퓨터 메모리에 저장된 실제 값

컴퓨터에서 값에 대한 자리 표시자로 이름을 사용하는 방법에 대해 설명했습니다. 데이터 유형도 중요합니다. ActionScript에 변수를 만들 때 변수에 저장할 데이터의 특정 유형을 지정합니다. 이후부터 프로그램의 명령은 지정된 유형의 데이터만 변수에 저장할 수 있으며 해당 데이터 유형과 연관된 특정 문자를 사용하여 값을 조작할 수 있습니다. ActionScript에서 변수를 만들려면 또는 변수를 *선언*하려면 다음과 같이 `var` 명령문을 사용합니다.

```
var value1:Number;
```

이 경우 "Number"(ActionScript에 정의된 특정 데이터 유형) 데이터만 저장하는 `value1` 변수를 만듭니다. 또한 다음과 같이 변수에 값을 바로 저장할 수도 있습니다.

```
var value2:Number = 17;
```

Adobe Flash CS3 Professional에서는 또 다른 방법으로 변수를 선언할 수 있습니다. Stage에 텍스트 필드, 버튼 심볼 또는 무비 클립 심볼을 배치할 때 속성 관리자에서 인스턴스 이름을 지정할 수 있습니다. Flash에서 내부적으로 인스턴스 이름과 동일한 이름의 변수를 만들므로 ActionScript 코드에서 해당 Stage 항목을 참조할 때 이 변수를 사용할 수 있습니다. 예를 들어, Stage에 있는 무비 클립에 인스턴스 이름으로 `rocketShip`을 지정하면 ActionScript 코드에서 `rocketShip` 변수를 사용할 때마다 실제로 해당 무비 클립을 조작할 수 있습니다.

데이터 유형

ActionScript에는 변수의 데이터 유형으로 사용할 수 있는 여러 데이터 유형이 있습니다. 다음과 같은 데이터 유형을 "간단"하고 "기본적인" 데이터 유형으로 간주할 수 있습니다.

- String: 설명서의 텍스트 또는 이름과 같은 텍스트 값입니다.
- Numeric: ActionScript 3.0에는 숫자 데이터에 사용할 수 있는 세 가지 데이터 유형이 있습니다.
  - Number: 분수이거나 분수가 아닌 값을 비롯한 모든 숫자 값
  - int: 정수(분수가 아닌 모든 숫자)
  - uint: "부호 없는" 정수(음수가 될 수 없는 모든 숫자)
- Boolean: 스위치가 켜져 있는지 또는 두 값이 같은지 여부를 지정하는 true 또는 false 값

간단한 데이터 유형은 단일 숫자 또는 텍스트의 단일 시퀀스 같은 단일 정보를 나타냅니다. 그러나 ActionScript에 정의된 대부분의 데이터 유형은 그룹화된 값의 집합을 나타내기 때문에 복합 데이터 유형으로 설명될 수 있습니다. 예를 들어, Date 데이터 유형의 변수는 해당 시점을 의미하는 단일 값을 나타냅니다. 그러나 실제로 날짜 값은 모두 개별 숫자인 일, 월, 시, 분, 초 등의 여러 값으로 나타납니다. 우리는 날짜를 단일 값으로 간주하고 실제로 Date 변수를 만들어 날짜를 단일 값으로 취급할 수 있지만 컴퓨터 내부적으로는 날짜를 여러 값이 결합하여 하나의 날짜를 정의하는 집합으로 간주합니다.

대부분의 내장 데이터 유형 및 프로그래머가 정의한 데이터 유형은 복합 데이터 유형입니다. 널리 사용되는 복합 데이터 유형은 다음과 같습니다.

- MovieClip: 무비 클립 심볼
- TextField: 동적 또는 입력 텍스트 필드
- SimpleButton: 버튼 심볼
- Date: 해당 시점에 대한 정보(날짜 및 시간)

클래스와 객체는 데이터 유형에 대한 동의어로 자주 사용되는 단어입니다. *클래스*는 특정 데이터 유형의 모든 객체에 대한 템플릿과 같은 데이터 유형 정의이며 "Example 클래스를 정의한다는 것은 Example 데이터 유형의 모든 변수에 A, B 및 C 등의 특징이 있다고 말하는 것과 같습니다." 반면에 *객체*는 클래스의 실제 인스턴스로 데이터 유형이 MovieClip인 변수는 MovieClip 객체로 설명됩니다. 다음과 같이 동일한 변수를 각각 다르게 설명할 수 있습니다.

- `myVariable` 변수의 데이터 유형은 Number입니다.
- `myVariable` 변수는 Number 인스턴스입니다.
- `myVariable` 변수는 Number 객체입니다.
- `myVariable` 변수는 Number 클래스의 인스턴스입니다.

객체 다루기

ActionScript는 객체 지향 프로그래밍 언어입니다. 객체 지향 프로그래밍은 프로그래밍에 대한 접근 방법 중 하나로 객체를 사용하여 프로그램의 코드를 구성하는 방법일 뿐입니다.

이 설명서의 앞 부분에 컴퓨터 프로그램은 컴퓨터에서 수행되는 일련의 단계 또는 명령으로 설명되어 있습니다. 개념적인 면에서 컴퓨터 프로그램은 명령으로 구성된 하나의 긴 목록이라고 가정할 수 있습니다. 그러나 객체 지향 프로그래밍에서는 프로그램 명령이 여러 객체에 분배됩니다. 코드가 기능별로 그룹화되고 관련된 유형의 기능 또는 관련된 정보가 컨테이너 하나에 그룹화됩니다.

실제로 Flash에서 심볼을 사용하여 작업해 본 경험이 있으면 객체를 사용하여 작업을 수행할 준비가 되어 있는 것입니다. 사각형의 무비 클립 심볼을 정의하고 Stage에 복사본을 배치했다고 가정해 봅니다. ActionScript에서는 무비 클립 심볼도 객체이며 MovieClip 클래스의 인스턴스입니다.

무비 클립의 다양한 특징을 수정할 수 있습니다. 예를 들어, 무비 클립이 선택되어 있으면 속성 관리자에서 x 좌표를 변경하고 너비를 변경하고 색을 다양하게 조정(알파 또는 투명도 변경)하거나 그림자 필터를 적용하는 등 다양한 값을 변경할 수 있습니다. [자유 변형] 도구를 사용하여 사각형을 회전하는 것과 같이 다른 Flash 도구를 사용하여 보다 많은 사항을 변경할 수 있습니다. Flash 제작 환경에서 무비 클립 심볼을 수정하기 위해 수행할 수 있는 여러 작업을 ActionScript에서는 MovieClip 객체라는 단일 묶음에 함께 포함된 여러 데이터를 변경하여 수행할 수 있습니다.

ActionScript 객체 지향 프로그래밍에는 모든 클래스에 포함될 수 있는 다음과 같은 세 가지 특성이 있습니다.

- 속성
- 메서드
- 이벤트

프로그램에서 사용하는 데이터를 관리하고 수행할 작업과 순서를 결정하는 데 이러한 요소가 함께 사용됩니다.

속성

속성은 객체에 함께 묶여 있는 데이터 중 하나를 나타냅니다. Song 객체에는 `artist` 속성과 `title` 속성이 포함될 수 있으며 MovieClip 클래스에는 `rotation`, `x`, `width` 및 `alpha`와 같은 속성이 포함될 수 있습니다. 속성은 개별적인 변수처럼 사용합니다. 실제로 속성을 객체에 포함된 "자식" 변수로 간주할 수 있습니다.

속성을 사용하는 ActionScript 코드의 예제는 다음과 같습니다. 다음 코드 행은 `square` MovieClip을 x 좌표 100픽셀로 이동합니다.

```
square.x = 100;
```

다음 코드에서는 rotation 속성을 사용하여 `triangle` MovieClip의 회전과 일치하도록 `square` MovieClip을 회전합니다.

```
square.rotation = triangle.rotation;
```

다음 코드에서는 수평 배율을 변경하여 `square` MovieClip의 폭을 1.5배 넓힙니다.

```
square.scaleX = 1.5;
```

객체 이름으로 변수(`square`, `triangle`)를 사용하고 그 뒤에 마침표(.)와 속성 이름(`x`, `rotation`, `scaleX`)을 차례로 사용하는 것이 일반적인 구조입니다. *도트 연산자*인 마침표는 객체의 자식 요소 중 하나에 액세스한다는 것을 나타내는 데 사용됩니다. 컴퓨터 메모리에 있는 단일 값의 이름으로 "변수 이름-도트-속성 이름"의 전체 구조를 단일 변수와 같이 사용합니다.

메서드

*메서드*는 객체가 수행할 수 있는 작업입니다. 예를 들어, 타임라인의 애니메이션 및 여러 키 프레임을 사용하여 Flash에 무비 클립 심볼을 만든 경우 해당 무비 클립을 재생 또는 중지하거나 재생 헤드를 특정 프레임으로 이동하도록 지시할 수 있습니다.

다음 코드는 `shortFilm` MovieClip의 재생을 시작하도록 지시합니다.

```
shortFilm.play();
```

다음 행은 `shortFilm` MovieClip의 재생을 중지합니다. 이렇게 하면 비디오를 일시 정지하는 것과 같이 재생 헤드가 제자리에 멈춥니다.

```
shortFilm.stop();
```

다음 코드는 비디오를 되감는 것과 같이 `shortFilm` MovieClip의 재생 헤드를 Frame 1로 이동하고 재생을 중지합니다.

```
shortFilm.gotoAndStop(1);
```

속성에서와 같이 객체 이름(변수) 다음에 마침표를 사용한 다음 메서드 이름 뒤에 괄호를 입력하여 메서드에 액세스합니다. 괄호는 메서드를 *호출*하는 것 즉, 객체가 해당 작업을 수행하도록 지시하는 것을 나타내는 방법입니다. 작업을 수행하는 데 필요한 추가 정보를 표시하는 방법으로 괄호 안에 값 또는 변수가 지정되는 경우도 있습니다. 이러한 값을 메서드 *매개 변수*라고 합니다. 예를 들어, `gotoAndStop()` 메서드가 이동해야 할 프레임을 표시해야 하므로 괄호 안에 단일 매개 변수가 있어야 합니다. `play()` 및 `stop()`과 같은 기타 메서드는 이름 자체로도 그 기능을 충분히 설명할 수 있으므로 추가 정보가 필요하지 않습니다. 그러나 괄호는 사용해야 합니다.

속성 및 변수와 달리 메서드는 값 자리 표시자로 사용되지 않습니다. 그러나 일부 메서드는 계산을 수행하여 변수처럼 사용할 수 있는 결과를 반환할 수 있습니다. 예를 들어, Number 클래스의 `toString()` 메서드는 숫자 값을 텍스트 표현으로 변환합니다.

```
var numericData:Number = 9;
var textData:String = numericData.toString();
```

예를 들어, 스크린의 텍스트 필드에 Number 변수의 값을 표시하려면 `toString()` 메서드를 사용합니다. 스크린에 표시된 실제 텍스트 내용을 나타내는 TextField 클래스의 `text` 속성이 String으로 정의되어 텍스트 값만 포함할 수 있습니다. 다음 코드 행은 `numericData` 변수의 숫자 값을 텍스트로 변환한 다음 `calculatorDisplay`라는 TextField 객체에 저장하여 해당 텍스트가 스크린에 표시되도록 합니다.

```
calculatorDisplay.text = numericData.toString();
```

이벤트

이 설명서의 앞 부분에 컴퓨터 프로그램은 컴퓨터에서 단계별로 수행하는 일련의 명령으로 설명되어 있습니다. 일부 간단한 컴퓨터 프로그램에는 프로그램이 끝나는 시점에 컴퓨터에서 수행되는 몇 가지 단계만 포함되어 있습니다. 그러나 ActionScript 프로그램은 사용자 입력 등을 기다리면서 계속해서 실행되도록 설계되었습니다. 이벤트는 컴퓨터에서 언제 어떤 명령을 수행할지 결정하는 메커니즘입니다.

기본적으로 *이벤트*는 ActionScript에서 감지하고 반응할 수 있는 사건입니다. 사용자가 버튼을 클릭하거나 키보드에 있는 키를 누르는 것과 같은 사용자 상호 작용에 많은 이벤트가 연결되어 있지만 그렇지 않은 이벤트 유형도 있습니다. 예를 들어, ActionScript를 사용하여 외부 이미지를 로드하는 경우 이미지 로드가 끝나면 알려 주는 이벤트가 있습니다. 기본적으로 ActionScript 프로그램을 실행할 때 Adobe Flash Player는 특정 이벤트가 발생하기를 기다리다가 이벤트가 발생하면 해당 이벤트에 지정된 특정 ActionScript 코드를 실행합니다.

기본적 이벤트 처리

*이벤트 처리*는 특정 이벤트에 응답해야 하는 특정 작업을 지정하기 위한 방법입니다. 이벤트 처리를 수행하는 ActionScript 코드를 작성할 때 다음 세 가지 중요한 요소를 식별할 수 있어야 합니다.

- 이벤트 소스: 이벤트가 발생할 객체 예를 들어, 클릭할 버튼 또는 이미지를 로드할 Loader 객체 등이 이벤트 소스입니다. 이벤트 소스는 Flash Player에 의해 이벤트 대상으로 지정되는 객체(실제로 이벤트가 발생하는 객체)이므로 *이벤트 대상*이라고도 합니다.
- 이벤트: 발생할 사건 및 응답할 사건 많은 객체에서 여러 이벤트를 트리거하기 때문에 이를 식별하는 것이 중요합니다.
- 응답: 이벤트가 발생하면 실행할 단계

이벤트를 처리할 ActionScript 코드를 작성할 때마다 이러한 세 가지 요소가 포함되고 해당 코드는 다음과 같은 기본 구조를 따릅니다. 여기서 굵게 표시된 요소는 특정한 경우에 입력하는 자리 표시자입니다.

```
function eventResponse(eventObject:EventType):void
{
  // 이벤트에 응답하여 수행할 액션을 여기에 입력하십시오 .
}

eventSource.addEventListener(EventType.EVENT_NAME, eventResponse);
```

이 코드는 두 가지 작업을 수행하는데, 먼저 함수를 정의하여 이벤트에 대한 응답으로 수행하려는 작업을 지정할 수 있습니다. 그런 다음 소스 객체의 `addEventListener()` 메서드를 호출하여 지정된 이벤트에 대해 함수를 "제공함으로써" 이벤트가 발생되면 함수 작업이 수행되도록 합니다. 이러한 부분을 개별적으로 보다 자세히 고찰해 보겠습니다.

*함수*는 단축키 이름 같은 단일 이름으로 작업을 그룹화하여 수행할 수 있는 방법을 제공합니다. 함수는 특정 클래스와 연결될 필요가 없다는 점을 제외하면 메서드와 동일합니다. 실제로 메서드를 특정 클래스와 연결된 함수로 정의할 수 있습니다. 이벤트 처리를 위해 함수를 만드는 경우 함수에 대한 이름(이 경우 `eventResponse`)을 선택하고 매개 변수(이 예제에서는 `eventObject`)도 하나 지정해야 합니다. 함수 매개 변수를 지정하는 것은 변수를 선언하는 것과 같기 때문에 매개 변수의 데이터 유형도 지정해야 합니다. 각 이벤트에는 ActionScript 클래스가 정의되어 있으며, 함수 매개 변수에 지정하는 데이터 유형은 언제나 응답하려는 특정 이벤트와 연관된 클래스입니다. 마지막으로, 여는 중괄호와 닫는 중괄호({ ... }) 사이에 이벤트가 발생할 때 컴퓨터에서 수행하려는 명령을 입력합니다.

이벤트 처리 함수를 작성한 후 이벤트 발생 시 이벤트 처리 함수를 호출하도록 버튼 등의 이벤트 소스 객체(이벤트가 발생할 객체)에 알려야 합니다. 이렇게 하려면 이벤트 소스 객체의 `addEventListener()` 메서드를 호출합니다. 모든 이벤트 소스 객체에는 `addEventListener()` 메서드가 있습니다. `addEventListener()` 메서드에 다음과 같은 두 개의 매개 변수가 포함됩니다.

- 첫 번째, 응답할 특정 이벤트 이름입니다. 각 이벤트가 특정 클래스와 연결되어 있고 이 클래스에는 각 이벤트에 대해 미리 정의된 특수 값이 있습니다. 이러한 값 중에 이벤트의 고유 이름이 있는데 첫 번째 매개 변수에는 이 값을 사용해야 합니다.
- 두 번째, 이벤트 응답 함수 이름입니다. 함수 이름이 매개 변수로 전달되면 괄호 없이 작성됩니다.

이벤트 처리 프로세스 검토

다음은 이벤트 리스너를 만들 때 발생하는 프로세스의 단계별 설명입니다. 이 경우는 myButton이라는 객체를 클릭할 때 호출되는 리스너 함수를 만드는 예제입니다.

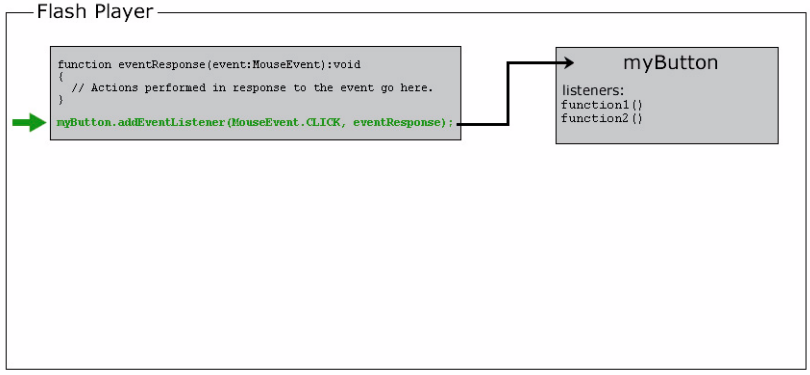
프로그래머가 작성하는 실제 코드는 다음과 같습니다.

```
function eventResponse(event:MouseEvent):void
{
  // 이벤트에 응답하여 수행할 액션을 여기에 입력하십시오 .
}

myButton.addEventListener(MouseEvent.CLICK, eventResponse);
```

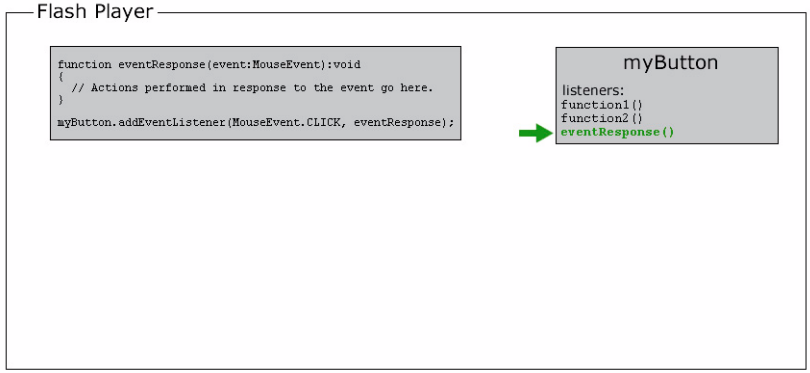
다음은 이 코드가 Flash Player에서 실행될 때 실제로 작동하는 방식을 나타낸 것입니다.

1. SWF 파일이 로드되면 Flash Player에서는 eventResponse()라는 함수의 존재 사실을 기억해 둡니다.

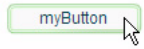
2. 그런 다음 Flash Player는 코드(특히 함수에 없는 코드 행)를 실행합니다. 이 경우에는 이벤트 소스 객체(myButton)에서 addEventListener() 메서드를 호출하고 eventResponse 함수를 매개 변수로 전달하여 하나의 코드 행만 실행합니다.



a. 내부적으로 myButton에는 각 해당 이벤트를 수신하는 함수 목록이 있으므로 해당 addEventListener() 메서드를 호출하면 myButton의 이벤트 리스너 목록에 eventResponse() 함수가 저장됩니다.

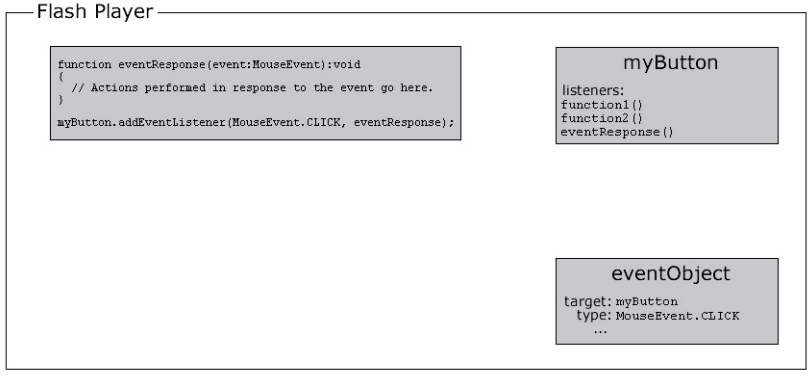


3. 어느 시점에서 사용자가 `myButton` 객체를 클릭하면 `click` 이벤트(코드에서 `MouseEvent.CLICK`으로 나타남)가 트리거됩니다.

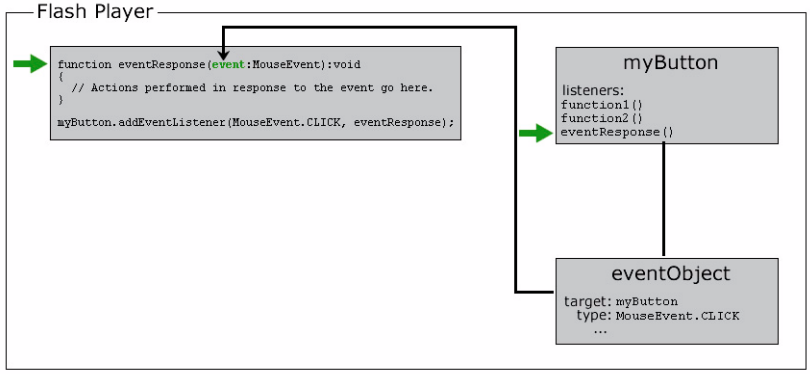


이때 다음과 같은 상황이 발생합니다.

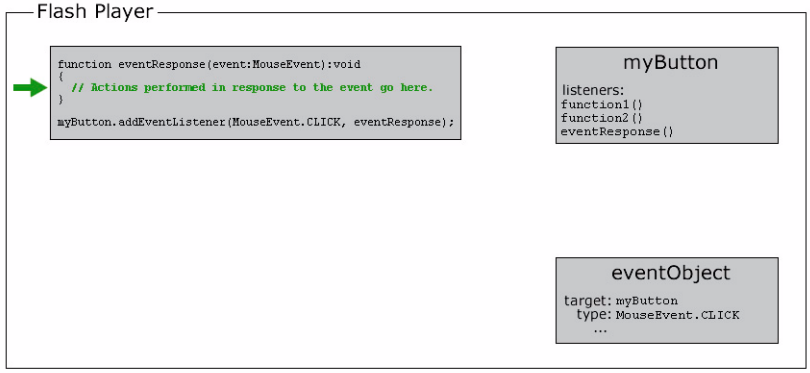
a. Flash Player는 해당 이벤트(이 예제에서는 MouseEvent)와 연관된 클래스의 인스턴스인 객체를 만듭니다. 여러 이벤트에서 이 객체는 Event 클래스의 인스턴스가 되며, 마우스 이벤트에서는 MouseEvent 인스턴스가, 기타 이벤트의 경우 해당 이벤트와 연관된 클래스의 인스턴스가 됩니다. 생성된 이 객체는 *이벤트 객체*라고 하며, 발생한 이벤트에 대한 고유 정보(이벤트 유형, 이벤트 발생 위치 등) 및 적용 가능한 기타 이벤트별 정보가 포함되어 있습니다.



b. 그런 다음 Flash Player는 `myButton`에 의해 저장된 이벤트 리스너 목록을 검토합니다. 이러한 함수를 하나씩 검토하면서 각 함수를 호출하고 이벤트 객체를 매개 변수로 함수에 전달합니다. `eventResponse()` 함수는 `myButton`의 리스너 중 하나이므로, Flash Player는 이 프로세스의 일부로서 `eventResponse()` 함수를 호출합니다.



c. `eventResponse()` 함수가 호출되면 해당 함수의 코드가 실행되어 특정 작업이 수행됩니다.



이벤트 처리 예제

다음에는 이벤트 처리 코드를 작성할 때 사용할 수 있는 일반적인 이벤트 요소 및 가능한 변형에 대해 설명하기 위한 보다 구체적인 예제가 나와 있습니다.

- 버튼을 클릭하여 현재 무비 클립 재생을 시작합니다. 다음 예제에서 playButton는 버튼의 인스턴스 이름이며 this는 "현재 객체"를 나타내는 특수 이름입니다.

```
this.stop();

function playMovie(event:MouseEvent):void
{
  this.play();
}

playButton.addEventListener(MouseEvent.CLICK, playMovie);
```

- 텍스트 필드에 입력된 내용을 검색합니다. 이 예제에서 entryText는 입력 텍스트 필드이며 outputText는 동적 텍스트 필드입니다.

```
function updateOutput(event:TextEvent):void
{
  var pressedKey:String = event.text;
  outputText.text = "You typed: " + pressedKey;
}

entryText.addEventListener(TextEvent.TEXT_INPUT, updateOutput);
```

- 버튼을 클릭하여 URL로 이동합니다. 이 경우 linkButton은 버튼의 인스턴스 이름입니다.

```
function gotoAdobeSite(event:MouseEvent):void
{
  var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
  navigateToURL(adobeURL);
}

linkButton.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

객체 인스턴스 만들기

ActionScript에서 객체를 사용하려면 먼저 객체가 있어야 합니다. 객체를 만드는 방법 중 하나는 변수를 선언하는 것입니다. 그러나 변수를 선언하면 컴퓨터 메모리에 빈 공간만 만들어 집니다. 변수에 실제 값을 지정해야 합니다. 즉, 객체를 사용하거나 조작하려면 먼저 객체를 만들고 변수에 해당 객체를 저장합니다. 객체를 만드는 과정을 객체 *인스턴스화*라고 합니다. 즉, 특정 클래스의 인스턴스를 만드는 것입니다.

객체 인스턴스를 간단히 만드는 방법은 ActionScript를 전혀 사용하지 않는 것입니다. Flash에서 Stage에 텍스트 필드, 버튼 심볼 또는 무비 클립 심볼을 배치하고 속성 관리자에서 인스턴스 이름을 지정하면, Flash는 인스턴스 이름과 동일한 이름의 변수를 자동으로 선언하고 객체 인스턴스를 만든 다음 변수에 해당 객체를 저장합니다. 마찬가지로 Adobe Flex Builder에서 MXML 태그를 코딩하거나 디자인 모드에서 편집기에 구성 요소를 배치하여 구성 요소를 Adobe Macromedia® MXML™로 만들고, MXML 마크업 또는 Flex 속성 보기에서 ID를 해당 구성 요소에 지정하면, ID는 ActionScript 변수의 이름이 되고 해당 구성 요소의 인스턴스가 만들어져 변수에 저장됩니다.

그러나 객체를 시각적으로 만들고 싶지 않을 때도 있습니다. ActionScript만 사용하여 객체 인스턴스를 만들 수 있는 여러 가지 방법이 있습니다. 먼저 ActionScript 코드에 직접 작성된 값인 *리터럴 표현식*을 사용하여 여러 ActionScript 데이터 유형의 인스턴스를 만들 수 있습니다. 다음에 몇 가지 예제가 나와 있습니다.

- 직접 숫자를 입력하는 리터럴 숫자 값

```
var someNumber:Number = 17.239;
var someNegativeInteger:int = -53;
var someUint:uint = 22;
```

- 텍스트가 따옴표로 둘러싸인 리터럴 문자열 값

```
var firstName:String = "George";
var soliloquy:String = "To be or not to be, that is the question...";
```

- `true` 또는 `false` 리터럴 값을 사용하는 리터럴 부울 값

```
var niceWeather:Boolean = true;
var playingOutside:Boolean = false;
```

- XML을 직접 입력하는 리터럴 XML 값

```
var employee:XML = <employee>
    <firstName>Harold</firstName>
    <lastName>Webster</lastName>
  </employee>;
```

또한 ActionScript에서 Array, RegExp, Object 및 Function 데이터 유형의 리터럴 표현식을 정의합니다. 이 클래스에 대한 자세한 내용은 211페이지의 “배열을 사용한 작업”, 269페이지의 “일반 표현식 사용” 및 92페이지의 “Object 데이터 유형”을 참조하십시오.

기타 모든 데이터 유형에 대해 객체 인스턴스를 만들려면 다음과 같이 `new` 연산자 클래스 이름과 함께 사용합니다.

```
var raceCar:MovieClip = new MovieClip();
var birthday:Date = new Date(2006, 7, 9);
```

new 연산자를 사용하여 객체를 만드는 것을 종종 "클래스 생성자를 호출한다"고 말합니다. *생성자*는 클래스 인스턴스를 만드는 과정 중 호출되는 특수 메서드입니다. 이러한 방법으로 인스턴스를 만드는 경우에는 클래스 이름 뒤에 괄호를 입력하고 필요하면 매개 변수 값을 지정하기도 합니다. 메서드를 호출할 때도 이와 같은 두 가지 작업을 수행합니다.

예

리터럴 표현식을 사용하여 인스턴스를 작성할 수 있는 데이터 유형의 경우에도 new 연산자를 사용하여 객체 인스턴스를 만들 수 있습니다. 예를 들어, 다음 두 코드 행은 정확하게 동일한 작업을 수행합니다.

```
var someNumber:Number = 6.33;
var someNumber:Number = new Number(6.33);
```

객체를 만드는 방법인 new *ClassName*()에 익숙해져야 합니다. 시각적으로 표시되지 않아 Flex Builder MXML 편집기의 디자인 모드 또는 Flash Stage에 항목을 배치하여 인스턴스를 만들 수 없는 ActionScript 데이터 유형의 인스턴스를 만들어야 하는 경우, ActionScript에서 new 연산자를 사용하여 직접 객체를 만들어야만 필요한 인스턴스를 만들 수 있습니다.

특히 Flash에서는 라이브러리에 정의되어 있지만 Stage에 배치되지 않은 무비 클립 심볼의 인스턴스를 만들 때 new 연산자를 사용합니다. 자세한 내용은 467페이지의 "ActionScript를 사용하여 MovieClip 객체 만들기"를 참조하십시오.

일반적 프로그램 요소

변수를 선언하고 객체 인스턴스를 만들고 객체의 속성 및 메서드를 사용하여 객체를 조작하는 기능 외에 ActionScript 프로그램을 만들 때 사용하는 다른 구성 단위가 몇 가지 있습니다.

연산자

*연산자*는 계산하는 데 사용되는 특수 기호이며 단어인 경우도 있습니다. 대부분 수학 연산에 사용되며 각 연산 값을 서로 비교하는 경우에도 사용됩니다. 일반적으로 연산자는 하나 이상의 값을 사용하여 하나의 결과로 "산출"합니다. 예를 들면 다음과 같습니다.

- 더하기 연산자(+)는 다음과 같이 두 개의 값을 서로 더하여 단일 숫자로 결과를 산출합니다.

  ```
  var sum:Number = 23 + 32;
  ```

- 곱하기 연산자(*)는 하나의 값을 다른 값으로 곱하여 단일 숫자로 결과를 산출합니다.

  ```
  var energy:Number = mass * speedOfLight * speedOfLight;
  ```

- 항등 연산자(==)는 두 개의 값이 서로 같은지 비교하여 true 또는 false(Boolean) 단일 값으로 결과를 산출합니다.

  ```
  if (dayOfWeek == "Wednesday")
  {
  ```

```
    takeOutTrash();
}
```

위의 예제에서와 같이 항등 연산자와 기타 "비교" 연산자는 특정 명령을 수행할 것인지 여부를 결정하기 위해 대부분의 경우 `if` 문과 함께 사용됩니다.

연산자 사용에 대한 예제 및 자세한 내용은 104페이지의 "연산자"를 참조하십시오.

주석

ActionScript를 작성할 때 특정 코드 행이 작동하는 방법 또는 특정 항목을 선택한 이유를 설명하는 메모를 남기려는 경우가 있습니다. *코드 주석*은 코드에서 무시할 수 있는 텍스트를 작성하는 데 사용할 수 있는 도구입니다. ActionScript에는 다음과 같은 두 가지 주석이 있습니다.

- 한 줄 주석: 한 줄 주석은 줄에서 원하는 위치에 두 개의 슬래시를 배치하여 지정합니다. 다음과 같이 슬래시 뒤부터 해당 줄이 끝나는 위치까지의 모든 내용은 컴퓨터에서 무시됩니다.

  ```
  // 이 부분은 주석이므로 컴퓨터에서 무시됩니다 .
  var age:Number = 10; // 기본적으로 나이를 10 으로 설정합니다 .
  ```

- 여러 줄 주석: 여러 줄 주석에는 주석 시작 표시자(/*) 뒤에 주석 내용이 포함된 다음 주석 끝 표시자(*/)가 포함됩니다. 다음과 같이 주석 범위의 줄 수에 관계없이 시작과 끝 표시자 사이의 모든 내용은 컴퓨터에서 무시됩니다.

  ```
  /*
  This might be a really long description, perhaps describing what
  a particular function is used for or explaining a section of code.

  In any case, these lines are all ignored by the computer.
  */
  ```

주석의 또 다른 일반적인 용도는 하나 이상의 코드 행을 일시적으로 "해제"하는 것입니다. 예를 들어, 작업을 수행하는 다른 방법을 테스트하거나 특정 ActionScript 코드가 예상대로 작동되지 않는 이유를 파악하려고 하는 경우 이렇게 주석을 사용합니다.

흐름 제어

프로그램에서 특정 작업을 반복하거나 특정 작업만 수행하거나 특정 조건에 따라 대체 작업을 수행하려는 경우가 자주 있습니다. *흐름 제어*는 수행되는 작업을 제어합니다. ActionScript에서 사용할 수 있는 여러 가지 흐름 제어 요소는 다음과 같습니다.

- 함수: 함수는 단축키와 같은 역할을 하며 단일 이름으로 일련의 작업을 그룹화하여 계산을 수행하는 데 사용할 수 있습니다. 특히 함수는 이벤트를 처리하는 데 중요한 역할을 하지만 일련의 명령을 그룹화하는 일반적인 도구로도 사용됩니다. 함수에 대한 자세한 내용은 117페이지의 "함수"를 참조하십시오.

- 루프: 루프 구조에서는 특정 횟수 또는 일부 조건이 변경될 때까지 컴퓨터에서 수행할 명령 집합을 지정할 수 있습니다. 주로 루프는 컴퓨터에서 루프를 통해 작업할 때마다 값이 변하는 변수를 사용하여 여러 관련 항목을 조작하는 데 사용됩니다. 루프에 대한 자세한 내용은 114페이지의 "반복"을 참조하십시오.
- 조건문: 조건문에서는 특정 환경에서만 수행되는 특정 명령을 지정하는 방법을 제공하거나 다른 조건에 맞는 대체 명령을 제공합니다. 가장 일반적인 조건문의 유형은 `if` 문입니다. `if` 문은 괄호 안에 있는 값 또는 표현식을 검사합니다. 값이 `true`이면 중괄호 안의 코드 행이 실행되고 그렇지 않으면 무시됩니다. 예를 들면 다음과 같습니다.

  ```
  if (age < 20)
  {
    // 십대와 관련된 특별한 내용
  }
  ```

  `if` 문과 함께 `else` 문을 사용하면 조건이 `true`가 아닌 경우 수행할 다른 명령을 지정할 수 있습니다.

  ```
  if (username == "admin")
  {
    // 추가 옵션 표시와 같은 관리자 전용 작업을 수행합니다 .
  }
  else
  {
    // 관리자 전용 작업을 수행합니다 .
  }
  ```

  조건문에 대한 자세한 내용은 111페이지의 "조건문"을 참조하십시오.

예제: 애니메이션 포트폴리오 작업

이 예제는 ActionScript를 처음 접하는 사용자에게 각 ActionScript 조각을 모아 완전한 하나의 응용 프로그램(ActionScript가 많이 사용되지 않은 경우)을 만드는 방법을 보여 주기 위한 것입니다. 애니메이션 포트폴리오 작업은 기존의 선형 애니메이션(예: 한 클라이언트용으로 만든 작업)을 온라인 포트폴리오에 통합하기 위해 적절한 약간의 대화형 요소를 추가하는 방법을 보여 주는 예제입니다. 애니메이션에 추가할 대화형 비헤이비어에는 보는 사람이 클릭할 수 있는 두 개의 버튼 즉, 애니메이션을 시작하는 버튼 및 별도의 URL(예: 포트폴리오 메뉴 또는 제작자의 홈 페이지)로 이동하는 버튼이 포함됩니다.

이 작업은 다음 주요 부분으로 나눌 수 있습니다.

1. ActionScript 및 대화형 요소 추가를 위한 FLA 파일 준비
2. 버튼 만들기 및 추가
3. ActionScript 코드 작성
4. 응용 프로그램 테스트

대화형 요소 추가 준비

대화형 요소를 애니메이션에 추가하기 전에 FLA 파일을 설정하여 새 내용을 추가할 위치를 만들어 놓는 것이 좋습니다. 즉, 스테이지에 버튼을 배치할 실제 공간을 만들고, 서로 다른 항목들을 개별적으로 배치할 "공간"도 FLA 파일에 만듭니다.

대화형 요소를 추가하기 위해 FLA를 설정하려면:

1. 대화형 요소를 추가할 대상 선형 애니메이션이 없는 경우, 단일 모션 트윈이나 모양 트윈 등 간단한 애니메이션이 포함된 새 FLA 파일을 만듭니다. 애니메이션이 있을 때는 프로젝트에 표시할 애니메이션이 포함된 FLA 파일을 열어 새 이름으로 저장하여 새 작업 파일을 만듭니다.
2. 두 개의 버튼(애니메이션 시작 버튼 및 제작자의 포트폴리오나 홈 페이지로 연결하는 버튼)을 배치할 화면 내의 위치를 결정합니다. 필요한 경우, 스테이지에서 공간을 없애거나 이 새 내용을 배치할 공간을 추가합니다. 애니메이션에 스플래쉬 화면이 없는 경우, 첫 번째 프레임에 스플래쉬 화면을 만들 수 있으며 애니메이션을 이동하여 프레임 2 이후에서 시작되도록 할 수도 있습니다.
3. 타임라인에 있는 다른 레이어 위에 새 레이어를 추가한 다음 이름을 **buttons**로 변경합니다. 이 레이어에 버튼을 추가하게 됩니다.
4. 이 buttons 레이어 위에 새 레이어를 추가한 다음 이름을 **actions**로 지정합니다. 이 레이어에서 ActionScript 코드를 사용자 응용 프로그램에 추가하게 됩니다.

버튼 만들기 및 추가

다음에는 대화형 응용 프로그램의 중심이 될 버튼을 실제로 만들어 배치해야 합니다.

버튼을 만들어 FLA에 추가하려면:

1. 드로잉 도구를 사용하여 buttons 레이어에 첫 번째 버튼("재생" 버튼) 모양을 만듭니다. 예를 들어, 수평 타원형을 그려 그 위에 텍스트를 표시할 수 있습니다.
2. [선택 도구]를 사용하여 이 버튼의 그래픽 부분을 모두 선택합니다.
3. 주 메뉴에서 [수정] > [심볼로 변환]을 선택합니다.
4. 표시되는 대화 상자에서 심볼 유형으로 [버튼]을 선택하고 심볼의 이름을 지정한 후 [확인]을 클릭합니다.
5. 이 버튼을 선택한 다음, 속성 관리자에서 버튼의 인스턴스 이름을 `playButton`으로 지정합니다.
6. 1-5단계를 반복하여 제작자의 홈 페이지로 연결되는 버튼을 만들고 이 버튼의 이름을 `homeButton`으로 지정합니다.

코드 작성

이 응용 프로그램의 ActionScript 코드는 동일한 위치에 배치되지만, 세 개의 기능 집합으로 나눌 수 있습니다. 코드가 수행할 세 가지 기능은 다음과 같습니다.

- SWF 파일 로드 즉시(재생 헤드가 프레임 1에 진입 시) 재생 헤드 중지
- 사용자가 재생 버튼 클릭 시 이벤트를 수신하여 SWF 파일 재생 시작
- 사용자가 제작자 홈 페이지 버튼 클릭 시 이벤트를 수신하여 브라우저를 해당 URL로 보냄

재생 헤드가 프레임 1에 진입 시 재생 헤드를 중지하도록 코드를 작성하려면:

1. actions 레이어의 프레임 1에서 키프레임을 선택합니다.
2. [액션] 패널을 열려면 주 메뉴에서 [윈도우] > [액션]을 선택합니다.
3. [스크립트] 창에 다음 코드를 입력합니다.

```
stop();
```

재생 버튼 클릭 시 애니메이션이 시작되도록 코드를 작성하려면:

1. 앞 단계에서 입력한 코드 맨 끝에 빈 행을 두 개 삽입합니다.
2. 스크립트 아래쪽에 다음 코드를 입력합니다.

```
function startMovie(event:MouseEvent):void
{
  this.play();
}
```

   이 코드는 `startMovie()`라는 함수를 정의합니다. `startMovie()`가 호출되면 이 함수에 의해 기본 타임라인에서 재생이 시작됩니다.

3. 앞 단계에서 추가된 코드 다음 행에 다음 코드 행을 입력합니다.

```
playButton.addEventListener(MouseEvent.CLICK, startMovie);
```

   이 코드 행은 `startMovie()` 함수를 `playButton`의 `click` 이벤트에 대한 리스너로 등록합니다. 즉, `playButton`이라는 버튼이 클릭될 때마다 `startMovie()` 함수가 호출되도록 하는 것입니다.

홈 페이지 버튼 클릭 시 브라우에서 해당 URL로 이동하도록 코드를 작성하려면:

1. 앞 단계에서 입력한 코드 맨 끝에 빈 행을 두 개 삽입합니다.
2. 스크립트 아래쪽에 다음 코드를 입력합니다.

```
function gotoAuthorPage(event:MouseEvent):void
{
  var targetURL:URLRequest = new URLRequest("http://example.com/");
  navigateToURL(targetURL);
}
```

   이 코드는 `gotoAuthorPage()`라는 함수를 정의합니다. 이 함수는 http://example.com/이라는 URL을 나타내는 URLRequest 인스턴스를 만든 다음, 이 URL을 `navigateToURL()` 함수에 전달하여 사용자 브라우저에서 이 URL이 열리도록 합니다.

3. 앞 단계에서 추가된 코드 다음 행에 다음 코드 행을 입력합니다.

```
homeButton.addEventListener(MouseEvent.CLICK, gotoAuthorPage);
```

   이 코드 행은 `gotoAuthorPage()` 함수를 `homeButton`의 `click` 이벤트에 대한 리스너로 등록합니다. 즉, `homeButton`이라는 버튼이 클릭될 때마다 `gotoAuthorPage()` 함수가 호출되도록 하는 것입니다.

응용 프로그램 테스트

이제 응용 프로그램이 완전히 작동됩니다. 실제로 작동되는지 테스트해 보겠습니다.

응용 프로그램을 테스트하려면:

1. 주 메뉴에서 [컨트롤] > [무비 테스트]를 선택합니다. Flash에서 SWF 파일을 생성하여 이 파일을 Flash Player 윈도우에서 엽니다.
2. 두 버튼이 모두 의도한 대로 작동하는지 테스트합니다.
3. 버튼이 작동하지 않는 경우 다음 사항을 확인합니다.
   - 두 버튼의 인스턴스 이름이 각각 다릅니까?
   - `addEventListener()` 메서드 호출 시 두 버튼의 인스턴스 이름과 동일한 이름을 사용합니까?
   - `addEventListener()` 메서드 호출 시 올바른 이벤트 이름이 사용되었습니까?
   - 각 함수에 대해 올바른 매개 변수가 지정되었습니까? 두 함수 모두 MouseEvent 데이터 유형을 가진 단일 매개 변수가 지정되어야 합니다.

   위의 사항을 따르지 않았거나 이외의 실수가 있는 경우, [무비 테스트] 명령을 선택하거나 해당 버튼을 클릭하면 오류 메시지가 나타납니다. [컴파일러 오류] 패널에서 컴파일러 오류([무비 테스트]를 처음 선택할 때 발생하는 오류)가 있는지 확인하고, [출력] 패널에서 런타임 오류(SWF 재생 도중 예를 들어, 버튼을 클릭할 때 발생하는 오류)가 있는지 확인합니다.

ActionScript로 응용 프로그램 만들기

ActionScript를 작성하여 응용 프로그램을 만들려면 사용할 클래스 이름과 구문을 아는 것만으로는 부족합니다. 이 설명서에 있는 대부분의 정보는 구문 및 ActionScript 클래스 사용의 두 항목에 대한 내용인 반면 이 설명서에는 나오지 않는 내용 즉, ActionScript 작성에 사용할 수 있는 프로그램, ActionScript 코드를 구성하고 응용 프로그램에 포함하는 방법 및 ActionScript 응용 프로그램을 개발할 때 수행해야 하는 단계 등에 대한 정보가 필요한 경우도 있습니다.

코드 구성에 대한 옵션

ActionScript 3.0 코드를 사용하여 단순한 그래픽 애니메이션에서 복잡한 클라이언트 서버 트랜잭션 처리 시스템까지의 모든 영역에서 프로그램을 개발할 수 있습니다. 만드는 응용 프로그램의 유형에 따라 프로젝트에 ActionScript를 포함하는 여러 방법 중 하나 이상을 사용할 수 있습니다.

Flash 타임라인의 프레임에 코드 저장

Flash 제작 환경에서 타임라인의 모든 프레임에 ActionScript 코드를 추가할 수 있습니다. 무비를 재생하는 동안 재생 헤드가 해당 프레임에 진입하면 이 코드가 실행됩니다.

프레임에 ActionScript 코드를 추가하면 Flash 제작 도구에서 만들어진 응용 프로그램에 비헤이비어를 간단하게 추가할 수 있습니다. 기본 타임라인의 모든 프레임에 코드를 추가하거나 모든 MovieClip 심볼에 대한 타임라인의 모든 프레임에 코드를 추가할 수 있습니다. 그러나 이와 같은 융통성을 갖추려면 대가를 치뤄야 합니다. 대규모 응용 프로그램을 만드는 경우 어떤 프레임에 어떤 스크립트가 포함되어 있는지 잊기 쉽습니다. 이로 인해 시간이 지날수록 응용 프로그램을 유지 관리하는 것이 보다 어려워질 수 있습니다.

많은 개발자들은 타임라인의 첫 번째 프레임 또는 Flash 문서의 특정 레이어에만 코드를 추가하여 Flash 제작 도구에서 ActionScript 코드 구성을 단순화합니다. 이렇게 하면 Flash FLA 파일에서 보다 쉽게 코드를 찾고 유지할 수 있습니다. 그러나 다른 Flash 프로젝트에서 동일한 코드를 사용하려면 코드를 복사하여 새 파일에 붙여 넣어야 합니다.

나중에 다른 Flash 프로젝트에서 ActionScript 코드를 사용하려면 외부 ActionScript 파일(확장명이 .as인 텍스트 파일)에 코드를 저장합니다.

ActionScript 파일에 코드 저장

프로젝트에 중요한 ActionScript 코드가 포함되는 경우 코드를 구성하는 가장 좋은 방법은 별도의 ActionScript 소스 파일(확장명이 .as인 텍스트 파일)에 코드를 구성하는 것입니다. 응용 프로그램에서 파일을 사용할 용도에 따라 둘 중 한 가지 방법으로 ActionScript 파일을 구성할 수 있습니다.

- 구조화되지 않은 ActionScript 코드: 타임라인 스크립트, MXML 파일 등에 직접 입력된 것 처럼 작성된 ActionScript 코드 행(명령문 또는 함수 정의 포함)입니다.

  ActionScript의 `include` 문 또는 Adobe Flex MXML의 `<mx:Script>` 태그를 사용하여 이러한 방식으로 작성된 ActionScript에 액세스할 수 있습니다. ActionScript의 `include` 문을 사용하면 특정 위치 및 스크립트에 지정된 범위 내에 외부 ActionScript 파일의 내용이 직접 입력한 것처럼 삽입됩니다. Flex MXML 언어에서 `<mx:Script>` 태그를 사용하면 응용 프로그램의 해당 위치에 로드할 외부 ActionScript 파일을 식별하는 소스 특성을 지정할 수 있습니다. 예를 들어, 다음 태그는 Box.as라는 외부 ActionScript 파일을 로드합니다.

  ```
  <mx:Script source="Box.as" />
  ```

- ActionScript 클래스 정의: 클래스의 메서드 및 속성 정의를 포함하는 ActionScript 클래스의 정의입니다.

  클래스를 정의하는 경우 모든 내장 ActionScript 클래스를 사용할 때와 동일하게 클래스의 인스턴스를 만들고 해당 속성, 메서드 및 이벤트를 사용하여 클래스에 있는 ActionScript 코드에 액세스할 수 있습니다. 이를 위해 다음과 같은 두 가지 과정을 수행해야 합니다.

  - ActionScript 컴파일러에서 해당 이름을 찾을 수 있도록 `import` 문을 사용하여 클래스의 전체 이름을 지정합니다. 예를 들어, ActionScript에 MovieClip 클래스를 사용하려면 먼저 패키지와 클래스를 포함하는 전체 이름을 사용하여 해당 클래스를 가져와야 합니다.

    ```
    import flash.display.MovieClip;
    ```

    또는 MovieClip 클래스가 포함된 패키지를 가져올 수 있습니다. 이는 패키지의 각 클래스에 대해 별도의 `import` 문을 작성하는 것과 동일합니다.

    ```
    import flash.display.*;
    ```

    코드에서 클래스를 참조하는 경우 클래스를 반드시 가져와야 하는 규칙의 유일한 예외는 최상위 클래스이며 이러한 클래스는 패키지에 정의되지 않습니다.

예외	Flash에서 타임라인의 프레임에 첨부된 스크립트의 경우 flash.* 패키지의 내장 클래스를 자동으로 가져옵니다. 그러나 직접 클래스를 작성하거나 Flash 제작 구성 요소(fl.* 패키지)로 작업하거나 Flex에서 작업하는 경우, 해당 클래스의 인스턴스를 만드는 코드를 작성하려면 모든 클래스를 명시적으로 가져와야 합니다.

- 클래스 이름을 참조하는 코드 즉, 일반적으로 데이터 유형으로 클래스를 사용하여 변수를 선언하고 변수에 저장할 클래스의 인스턴스를 만드는 코드를 작성합니다. ActionScript 코드에서 다른 클래스 이름을 참조하여 컴파일러에서 해당 클래스 정의를 로드하도록 지시합니다. 예를 들어, Box라는 외부 클래스가 지정되면 이 명령문이 Box 클래스의 새 인스턴스를 만듭니다.

  ```
  var smallBox:Box = new Box(10,20);
  ```

  컴파일러에서 Box 클래스에 대한 참조를 처음 발견한 경우 로드된 소스 코드를 검색하여 Box 클래스 정의를 찾습니다.

올바른 도구 선택

사용할 수 있는 리소스 및 프로젝트의 필요에 따라 ActionScript 코드를 작성하고 편집할 때 여러 도구 중 하나를 사용하거나 여러 도구를 서로 결합하여 사용할 수 있습니다.

Flash 제작 도구

Adobe Flash CS3 Professional에는 그래픽 및 애니메이션을 만드는 기능 외에도 FLA 파일 또는 외부 ActionScript 전용 파일의 요소에 연결된 ActionScript 코드로 작업할 수 있는 도구가 포함되어 있습니다. Flash 제작 도구는 중요한 애니메이션이나 비디오가 포함된 프로젝트, 대부분의 그래픽 에셋을 만들려는 프로젝트에 적합하며 특히 사용자 상호 작용을 최소화하는 프로젝트 또는 ActionScript가 필요한 기능을 구현하는 프로젝트에 적합합니다. ActionScript 프로젝트를 개발하는 데 Flash 제작 도구를 사용할 수 있는 또 다른 경우는 동일한 응용 프로그램에서 시각적 에셋을 만들고 코드를 작성하려는 경우입니다. 미리 만들어진 사용자 인터페이스 구성 요소를 사용하려고 하나 SWF의 크기를 줄이거나 시각적 스키닝을 쉽게 하는 것이 프로젝트에서 더 중요한 경우에도 Flash 제작 도구를 사용할 수 있습니다.

Adobe Flash CS3 Professional에는 다음과 같이 ActionScript 코드 작성을 위한 두 가지 도구가 포함되어 있습니다.

- [액션] 패널: FLA 파일에서 작업할 때 사용할 수 있으며 이 패널을 사용하면 타임라인의 프레임에 첨부된 ActionScript 코드를 작성할 수 있습니다.
- [스크립트] 윈도우: [스크립트] 윈도우는 ActionScript(.as) 코드 파일을 사용하여 작업하는 전용 텍스트 편집기입니다.

Flex Builder

Adobe Flex Builder는 Flex 프레임워크를 사용하여 프로젝트를 만들 때 가장 유용한 도구입니다. Flex Builder에는 시각적 레이아웃 및 MXML 편집 도구 외에도 완전한 기능의 ActionScript 편집기가 포함되어 있으므로 Flex 또는 ActionScript 전용 프로젝트를 만드는 데 사용할 수 있습니다. Flex 응용 프로그램에는 외부 데이터 소스를 사용하여 작업하고 외부 데이터를 사용자 인터페이스 요소에 연결할 수 있는 내장 메커니즘, 유연한 동적 레이아웃 제어 및 미리 만든 사용자 인터페이스 제어 등의 여러 가지 이점이 있습니다. 그러나 이러한 기능을 제공하려면 코드를 추가해야 하기 때문에 Flex 응용 프로그램에서 SWF 파일 크기가 커질 수 있으며 Flash 응용 프로그램처럼 쉽게 외관을 완전히 변경할 수 없습니다.

Flex를 사용하여 기능이 완전한 데이터 기반의 풍부한 인터넷 응용 프로그램을 만드는 경우 단일 도구 내에서 ActionScript 코드와 MXML 코드를 편집하고 응용 프로그램을 시각적으로 배치하려면 Flex Builder를 사용하십시오.

타사 ActionScript 편집기

ActionScript(.as) 파일은 간단한 텍스트 파일로 저장되기 때문에 ActionScript 파일을 작성하는 데 일반 텍스트 파일을 편집할 수 있는 모든 프로그램을 사용할 수 있습니다. Adobe ActionScript 제품 외에도 ActionScript 전용 기능이 포함된 여러 타사 텍스트 편집 프로그램이 있습니다. 모든 텍스트 편집기 프로그램을 사용하여 MXML 파일 또는 ActionScript 클래스를 작성할 수 있습니다. 그런 다음 Flex 프레임워크 클래스 및 Flex 컴파일러가 포함된 Flex SDK를 사용하여 해당 파일의 SWF 응용 프로그램(Flex 또는 ActionScript 전용 응용 프로그램)을 만들 수 있습니다. 또한 많은 개발자가 ActionScript 클래스를 작성할 수 있는 타사 ActionScript 편집기를 그래픽 내용을 만들 수 있는 Flash 제작 도구와 함께 사용하기도 합니다.

다음과 같은 경우 타사 ActionScript 편집기를 사용하기로 결정할 수 있습니다.

- Flash에서 시각적 요소를 디자인하면서 별도의 프로그램에 ActionScript 코드를 작성하려는 경우
- 다른 프로그래밍 언어로 응용 프로그램을 작성하거나 HTML 페이지를 만드는 것과 같이 비ActionScript 프로그래밍을 위한 응용 프로그램을 사용하고 있으며 ActionScript 코딩에도 동일한 응용 프로그램을 사용하려는 경우
- 리소스를 많이 차지하는 Flash 또는 Flex Builder 대신 Flex SDK를 사용하여 ActionScript 전용 또는 Flex 프로젝트를 만들려는 경우

ActionScript 전용 기능을 지원하는 주요 코드 편집기를 몇 가지 소개하면 다음과 같습니다.

- Adobe Dreamweaver® CS3
- ASDT
- FDT
- FlashDevelop
- PrimalScript

- SE|PY
- XCode(ActionScript 템플릿 및 코드 힌트 파일 포함)

ActionScript 개발 프로세스

ActionScript 프로젝트의 규모에 관계없이 응용 프로그램 설계 및 개발에 프로세스를 사용하면 보다 효율적이고 효과적으로 작업을 수행할 수 있습니다. 다음 단계는 ActionScript 3.0을 사용하는 응용 프로그램을 만들기 위한 기본 개발 프로세스를 설명합니다.

1. 응용 프로그램을 설계합니다.

   응용 프로그램을 만들기 전에 특정 방식으로 해당 응용 프로그램을 기술해야 합니다.

2. ActionScript 3.0 코드를 구성합니다.

   Flash, Flex Builder, Dreamweaver 또는 텍스트 편집기를 사용하여 ActionScript 코드를 만들 수 있습니다.

3. Flash 또는 Flex 응용 프로그램을 만들어 코드를 실행합니다.

   Flash 제작 도구에서는 이 과정에 새 FLA 파일을 만들고 제작 설정을 설정하고 사용자 인터페이스 구성 요소를 응용 프로그램에 추가하며 ActionScript 코드를 참조하는 작업이 포함됩니다. Flex 개발 환경에서는 새 응용 프로그램 파일을 만드는 과정에 응용 프로그램을 정의하고 MXML을 사용하여 사용자 인터페이스 구성 요소를 추가하고 ActionScript 코드를 참조하는 작업이 포함됩니다.

4. ActionScript 응용 프로그램을 제작하고 테스트합니다.

   이 과정에는 Flash 제작 또는 Flex 개발 환경에서 응용 프로그램을 실행하는 작업이 포함되며 응용 프로그램이 의도한 대로 작동하는지 확인해야 합니다.

이러한 단계를 순서대로 수행하지 않아도 되며 다른 작업을 수행하기 전에 수행 중이던 단계를 끝내야 할 필요는 없습니다. 예를 들어, 1단계에서 응용 프로그램의 스크린을 설계한 다음 3단계에서 그래픽, 단추 등을 만듭니다. 그런 다음 2단계에서 ActionScript 코드를 작성하고 4단계에서 테스트합니다. 또는 응용 프로그램의 일부를 설계한 다음 버튼 또는 인터페이스 요소를 한 번에 하나씩 추가하고 각 요소에 대한 ActionScript를 작성하여 만든 응용 프로그램을 테스트합니다. 개발 프로세스의 네 단계에 따라 개발 작업을 진행하는 것이 좋지만 실제 개발 환경에서는 일반적으로 앞 단계 또는 뒤 단계로 적절하게 이동하여 작업하는 것이 보다 효과적입니다.

클래스 만들기

프로젝트에 사용할 클래스를 만드는 과정은 어려워 보일 수 있습니다. 그러나 클래스를 만드는 과정에서 보다 어려운 부분은 클래스를 설계하는 작업 즉, 클래스에 포함될 메서드, 속성 및 이벤트를 식별하는 작업입니다.

클래스 설계 전략

객체 지향 설계는 이 분야의 학술적 연구와 실무에 모든 경력을 바친 사람에게도 어렵게 느껴지는 복잡한 주제입니다. 그러나 전문가가 아니더라도 객체 지향 설계를 시작할 수 있는 몇 가지 방법이 제안되어 있습니다.

1. 이 클래스의 인스턴스가 응용 프로그램에서 수행할 역할에 대해 생각해 봅니다.
   일반적으로 객체는 다음과 같은 세 가지 역할 중 하나를 수행합니다.
   - 값 객체: 이러한 객체는 주로 데이터의 컨테이너 역할을 합니다. 즉, 객체에 여러 속성과 적은 수의 메서드(또는 전혀 없음)가 포함되어 있는 경우가 많습니다. 이러한 객체는 일반적으로 뮤직 플레이어 응용 프로그램의 Song 클래스(실제 노래 한 곡 표현) 또는 Playlist 클래스(개념적인 노래 그룹 표현)처럼 명확하게 정의된 항목을 코드로 표현합니다.
   - 표시 객체: 실제로 스크린에 나타나는 객체입니다. 예를 들어, 드롭 다운 목록 또는 상태 판독과 같은 사용자 인터페이스 요소, 비디오 게임의 내용과 같은 그래픽 요소 등이 있습니다.
   - 응용 프로그램 구조: 이러한 객체는 응용 프로그램에서 수행하는 처리 또는 논리에서 광범위한 지원 역할을 수행합니다. 예를 들어, 바이올로지 시뮬레이션에서 특정 계산을 수행하는 객체, 뮤직 플레이어 응용 프로그램에서 다이얼 콘트롤과 볼륨 판독 간에 값의 동기화를 수행하는 객체, 비디오 게임에서 규칙을 관리하는 객체, 그리기 응용 프로그램에 저장된 그림을 로드하는 객체 등이 있습니다.
2. 클래스에 필요한 특정 기능을 결정합니다. 서로 다른 유형의 기능이 클래스의 여러 메서드로 표현되는 경우가 자주 있습니다.
3. 클래스를 값 객체로 사용하려는 경우 인스턴스에 포함할 데이터를 결정합니다. 이러한 항목은 속성으로 정의하기에 적합합니다.
4. 특정 프로젝트 전용으로 클래스를 설계 중이므로 응용 프로그램에 필요한 기능을 제공하는 것이 가장 중요합니다. 다음과 같은 질문의 답을 스스로 찾아 보는 것이 도움이 됩니다.
   - 응용 프로그램에서 저장, 추적 및 조작할 정보는 무엇입니까? 이 정보를 결정하면 원하는 모든 값 객체 및 속성을 식별하는 데 유용합니다.

- 수행해야 하는 작업에는 무엇이 있습니까? 예를 들어 응용 프로그램을 처음 로드하는 경우, 특정 버튼을 클릭한 경우, 무비 재생이 중지된 경우 등에 수행해야 하는 작업은 무엇입니까? 이러한 경우는 메서드로 정의하기에 적합하며 "작업"에서 개별 값만 변경하는 경우에는 메서드보다 속성으로 정의하는 것이 적합합니다.
- 지정된 작업이 있는 경우 해당 작업을 수행하기 위해 클래스에 전달해야 할 정보는 무엇입니까? 이러한 정보는 메서드의 매개 변수가 됩니다.
- 응용 프로그램에서 작업을 진행할 때 응용 프로그램의 다른 부분에 전달해야 할 클래스의 변경 내용은 무엇입니까? 이러한 변경 내용은 이벤트로 정의하기에 적합합니다.

5. 추가하려는 기능은 정의되어 있지 않지만 필요한 객체와 유사한 기존 객체가 있는 경우 하위 클래스를 만드는 것을 고려해 보십시오. 하위 클래스는 클래스 자체 기능을 정의하기 보다는 기존 클래스의 기능을 기반으로 하는 클래스입니다. 예를 들어, 스크린에서 시각적 객체로 사용할 클래스를 만들려는 경우 Sprite 또는 MovieClip과 같은 기존 표시 객체 중 하나의 비헤이비어를 클래스에 대한 기준으로 사용할 수 있습니다. 이 경우, *기본 클래스*는 MovieClip 또는 Sprite이며 사용자 정의 클래스는 기본 클래스를 확장합니다. 하위 클래스를 만드는 방법에 대한 자세한 내용은 154페이지의 "상속"을 참조하십시오.

클래스의 코드 작성

클래스 설계 계획이 있는 경우 또는 최소한 클래스에서 추적해야 할 정보 및 클래스에서 수행해야 할 작업을 생각해 둔 경우 클래스를 작성하는 실제 구문은 매우 간단합니다.

ActionScript 클래스를 만드는 최소 단계는 다음과 같습니다.

1. Flex Builder 또는 Flash와 같은 ActionScript 전용 프로그램, Dreamweaver와 같은 일반 프로그래밍 도구 또는 일반 텍스트 문서로 작업할 수 있는 모든 프로그램에서 새 텍스트 문서를 엽니다.
2. `class` 문을 입력하여 클래스 이름을 정의합니다. 이를 수행하려면 `public class`를 입력한 다음 클래스 내용(메서드 및 속성 정의)을 둘러싸는 열기 및 닫기 중괄호가 뒤에 오는 클래스 이름을 입력합니다. 예를 들면 다음과 같습니다.

   ```
   public class MyClass
   {
   }
   ```

   `public`은 다른 모든 코드에서 클래스에 액세스할 수 있다는 것을 나타냅니다. 다른 예제에 대한 자세한 내용은 137페이지의 "액세스 제어 네임스페이스 특성"을 참조하십시오.
3. package 문을 입력하고 찾을 패키지의 이름을 지정합니다. 구문은 `package` 다음에 전체 패키지 이름이 사용되고 `class` 문 블록을 둘러싸는 열기 및 닫기 중괄호가 사용됩니다. 예를 들어, 이전 단계의 코드를 다음과 같이 변경합니다.

   ```
   package mypackage
   {
     public class MyClass
   ```

```
    {
    }
}
```

4. 클래스 본문 내의 var 문을 사용하여 클래스에 각 속성을 정의합니다. 구문은 public 수정자를 추가하여 모든 변수를 선언할 때 사용한 구문과 동일합니다. 예를 들어, 클래스 정의의 여는 중괄호와 닫는 중괄호 사이에 줄을 추가하면 textVariable, numericVariable 및 dateVariable 속성이 만들어집니다.

```
public var textVariable:String = "some default value";
public var numericVariable:Number = 17;
public var dateVariable:Date;
```

5. 함수를 정의하는 데 사용된 구문과 동일한 구문을 사용하여 클래스에 각 메서드를 정의합니다. 예를 들면 다음과 같습니다.

   - myMethod() 메서드를 만들려면 다음을 입력합니다.

     ```
     public function myMethod(param1:String, param2:Number):void
     {
       // 매개 변수 작업을 합니다 .
     }
     ```

   - 클래스의 인스턴스를 만드는 과정 중에 호출되는 특수 메서드인 생성자를 만들려면 클래스 이름과 정확하게 일치하는 메서드 이름을 만듭니다.

     ```
     public function MyClass()
     {
       // 작업을 수행하여 속성의 초기값을 설정합니다 .
       // 그렇지 않으면 객체를 설정합니다 .
       textVariable = "Hello there!";
       dateVariable = new Date(2001, 5, 11);
     }
     ```

     클래스에 생성자 메서드가 포함되어 있지 않으면 컴파일러에서 클래스에 매개 변수와 명령문이 없는 비어 있는 생성자를 자동으로 만듭니다.

정의할 수 있는 클래스 요소가 몇 가지 더 있으며 이러한 요소는 다른 요소보다 복잡합니다.

- *접근자*는 메서드와 속성 사이의 특수한 중간적 요소입니다. 코드를 작성하여 클래스를 정의할 때 메서드와 같은 접근자를 작성하면 속성을 정의한 후 단순히 값을 읽고 지정할 수 있었던 것에 비해 여러 가지 작업을 수행할 수 있습니다. 그러나, 클래스의 인스턴스를 만들 때는 접근자를 속성과 같이 처리합니다. 속성의 경우 이름만 사용하여 값을 읽거나 지정합니다. 자세한 내용은 145페이지의 “get 및 set 접근자 메서드”를 참조하십시오.
- ActionScript의 이벤트는 특정 구문을 사용하여 정의하지 않습니다. 대신 이벤트 리스너를 추적하여 리스너에게 이벤트를 알리도록 EventDispatcher 클래스 기능을 사용하여 클래스에 이벤트를 정의합니다. 클래스에서 이벤트 만들기에 대한 자세한 내용은 295페이지의 제10장, “이벤트 처리”를 참조하십시오.

클래스 구성에 대한 제안

이전 버전의 ActionScript와 달리 ActionScript 3.0에는 파일당 하나의 클래스만 사용하도록 제한하는 제한 사항이 없습니다. ActionScript 3.0을 사용하면 하나의 .as 파일에 두 개 이상의 클래스에 대한 소스 코드를 저장할 수 있습니다. 여러 클래스를 하나의 소스 파일로 묶는 것이 보다 편리한 경우도 있지만 일반적으로 이는 좋지 않은 프로그래밍 방식으로 고려됩니다. 이에 대한 이유는 다음과 같습니다.

- 클래스가 하나의 큰 파일로 함께 묶여 있으면 개별 클래스를 다시 사용하기가 어렵습니다.
- 파일 이름이 클래스 이름과 동일하지 않으면 특정 클래스에 대한 소스 코드를 찾기가 어렵습니다.

이러한 이유로 항상 클래스 자체 파일에 각 개별 클래스의 소스 코드를 저장하고 파일 이름을 클래스와 동일한 이름으로 지정하는 것이 좋습니다.

예제: 기본 응용 프로그램 만들기

Flash, Flex Builder, Dreamweaver 또는 모든 텍스트 편집기를 사용하여 확장명이 .as인 외부 ActionScript 소스 파일을 만듭니다.

ActionScript 3.0은 Flash 제작 및 Flex Builder 도구를 포함하여 많은 응용 프로그램 개발 환경에 사용할 수 있습니다.

이 단원에서는 Flash 제작 도구 또는 Flex Builder 2 도구를 사용하여 간단한 ActionScript 3.0 응용 프로그램을 만들고 향상시킬 수 있는 단계에 대해 설명합니다. 작성할 응용 프로그램에서는 Flash 및 Flex 응용 프로그램에서 외부 ActionScript 3.0 클래스 파일을 사용하는 간단한 패턴을 제공합니다. 이 패턴은 설명서의 다른 샘플 응용 프로그램 모두에 적용됩니다.

ActionScript 응용 프로그램 설계

응용 프로그램을 만들려면 먼저 만들려는 응용 프로그램에 대한 구상이 있어야 합니다.

설계 표현은 응용 프로그램 이름 및 목적에 대한 간략한 설명과 같이 간단할 수도 있고, 여러 UML(Unified Modeling Language) 다이어그램이 포함된 필수 문서 세트와 같이 복잡할 수도 있습니다. 이 설명서에는 소프트웨어 설계 규칙에 대해 자세하게 설명되어 있지 않지만 ActionScript 응용 프로그램의 개발에서 응용 프로그램 설계는 필수 사항이라는 점을 명심해야 합니다.

ActionScript 응용 프로그램에 대한 첫 번째 예는 표준 “Hello World” 응용 프로그램이며 이에 대한 설계는 매우 간단합니다.

- 응용 프로그램을 HelloWorld라고 하겠습니다.
- “Hello World”가 포함된 단일 텍스트 필드가 표시됩니다.

- 쉽게 재사용할 수 있도록 Greeter라는 단일 객체 지향 클래스를 사용합니다. 이 클래스는 Flash 문서 또는 Flex 응용 프로그램에서 사용될 수 있습니다.
- 기본 버전의 응용 프로그램을 만든 후 사용자가 사용자 이름을 입력하면 응용 프로그램은 알려진 사용자 목록에서 사용자가 입력한 이름을 확인하도록 새 기능을 추가합니다.

간결한 정의가 완료되었으므로 응용 프로그램 작성을 시작할 수 있습니다.

HelloWorld 프로젝트 및 Greeter 클래스 만들기

Hello World 응용 프로그램의 설계 문장에 따르면 코드는 쉽게 재사용할 수 있어야 합니다. 이 목적에 부합하도록 응용 프로그램에서 Greeter라는 단일 객체 지향 클래스를 사용합니다. 이 클래스는 Flex Builder 또는 Flash 제작 도구에서 만든 응용 프로그램에 사용됩니다.

Flash 제작 도구에서 Greeter 클래스를 만들려면:

1. Flash 제작 도구에서 [파일] > [새로 만들기]를 선택합니다.
2. [새 문서] 대화 상자에서 [ActionScript 파일]을 선택하고 [확인]을 클릭합니다.

   새 ActionScript 편집 윈도우가 나타납니다.
3. [파일] > [저장]을 선택합니다. 응용 프로그램을 넣을 폴더를 선택하고 ActionScript 파일의 이름을 **Greeter.as**로 변경한 다음 [확인]을 클릭합니다.

   계속해서 55페이지의 “Greeter 클래스에 코드 추가”로 진행합니다.

Greeter 클래스에 코드 추가

Greeter 클래스는 HelloWorld 응용 프로그램에서 사용할 수 있는 객체인 `Greeter`를 정의합니다.

Greeter 클래스에 코드를 추가하려면:

1. 새 파일에 다음 코드를 입력합니다.

```
package
{
  public class Greeter
  {
    public function sayHello():String
    {
      var greeting:String;
      greeting = "Hello World!";
      return greeting;
    }
  }
}
```

   Greeter 클래스에는 주어진 사용자 이름에 대해 “Hello”라고 인사하는 문자열을 반환하는 단일 `sayHello()` 메서드가 포함되어 있습니다.

2. [파일] > [저장]을 선택하여 이 ActionScript 파일을 저장합니다.

이제 Flash 또는 Flex 응용 프로그램에서 Greeter 클래스를 사용할 수 있습니다.

ActionScript 코드를 사용하는 응용 프로그램 만들기

앞에서 만든 Greeter 클래스는 소프트웨어 기능의 독립된 집합을 정의하지만 완전한 응용 프로그램을 나타내는 것은 아닙니다. 이 클래스를 사용하려면 Flash 문서 또는 Flex 응용 프로그램을 만들어야 합니다.

HelloWorld 응용 프로그램에서 Greeter 클래스의 새 인스턴스를 만듭니다. 이 설명서에 Greeter 클래스를 응용 프로그램에 연결하는 방법이 설명되어 있습니다.

Flash 제작 도구를 사용하여 ActionScript 응용 프로그램을 만들려면:

1. [파일] > [새로 만들기]를 선택합니다.
2. [새 문서] 대화 상자에서 [Flash 문서]를 선택하고 [확인]을 클릭합니다.

   새 Flash 윈도우가 나타납니다.
3. [파일] > [저장]을 선택합니다. Greeter.as 클래스 파일이 들어 있는 동일한 폴더를 선택하여 Flash 문서를 **HelloWorld.fla**로 지정한 다음 [확인]을 클릭합니다.
4. [Flash 도구] 팔레트에서 [텍스트 도구]를 선택하여 스테이지에서 드래그하여 300픽셀(폭) x 100픽셀(높이) 정도 크기의 새 텍스트 필드를 만듭니다.
5. 스테이지의 새 텍스트 필드를 선택한 상태에서, [속성] 윈도우에 이 텍스트 필드의 인스턴스 이름으로 `mainText`를 입력합니다.
6. 기본 타임라인의 첫 번째 프레임을 클릭합니다.
7. [액션] 패널에 다음 스크립트를 입력합니다.

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("Bob");
```

8. 파일을 저장합니다.

계속해서 57페이지의 “ActionScript 응용 프로그램 제작 및 테스트”로 진행합니다.

ActionScript 응용 프로그램 제작 및 테스트

Software 개발은 반복되는 과정입니다. 코드를 작성하고 컴파일한 다음 해당 코드가 오류 없이 컴파일될 때까지 코드를 편집합니다. 컴파일된 응용 프로그램을 실행하고 테스트하여 의도했던 설계가 실현되었는지 확인하고, 원하는 설계가 실현되지 않은 경우 실현될 때까지 코드를 다시 편집합니다. Flash 및 Flex Builder 개발 환경에서는 응용 프로그램을 제작, 테스트 및 디버깅하는 여러 가지 방법을 제공합니다.

이 설명서에는 각 환경에서 HelloWorld 응용 프로그램을 테스트하기 위한 기본 단계가 설명되어 있습니다.

Flash 제작 도구를 사용하여 ActionScript 응용 프로그램을 제작 및 테스트하려면:

1. 응용 프로그램을 제작한 후 컴파일 오류가 있는지 확인합니다. Flash 제작 도구에서 [컨트롤] > [무비 테스트]를 선택하여 ActionScript 코드를 컴파일하고 HelloWorld 응용 프로그램을 실행합니다.
2. 응용 프로그램 테스트 시 [출력] 윈도우에 오류나 경고가 나타나면 HelloWorld.fla 파일이나 HelloWorld.as 파일에서 오류의 원인을 수정한 다음 응용 프로그램을 다시 테스트해 봅니다.
3. 컴파일 오류가 없는 경우에는 Hello World 응용 프로그램이 표시된 Flash Player 윈도우가 나타나고 “Hello, Bob”이라는 텍스트가 표시됩니다.

단순하지만 완전한 ActionScript 3.0 사용 객체 지향 응용 프로그램을 만들었습니다. 계속해서 57페이지의 “HelloWorld 응용 프로그램 개선”으로 진행합니다.

HelloWorld 응용 프로그램 개선

보다 재미있는 응용 프로그램을 만들기 위해 응용 프로그램에서 사용자에게 이름을 입력할 것을 요청하고 사용자가 입력한 이름이 유효한지 미리 정의된 이름 목록을 통해 확인하도록 합니다.

먼저 Greeter 클래스를 업데이트하고 새 기능을 추가합니다. 그런 다음 Flex 또는 Flash 응용 프로그램을 업데이트하여 새 기능을 사용합니다.

Greeter.as 파일을 업데이트하려면:

1. Greeter.as 파일을 엽니다.
2. 파일 내용을 다음과 같이 변경합니다. 새로 추가된 행과 변경된 행은 굵은체로 표시됩니다.

```
package
{
  public class Greeter
  {
    /**
     * Defines the names that should receive a proper greeting.
     */
```

```
    public static var validNames:Array = ["Sammy", "Frank", "Dean"];

    /**
     * Builds a greeting string using the given name.
     */
    public function sayHello(userName:String = ""):String
    {
      var greeting:String;
      if (userName == "")
      {
        greeting = "Hello. Please type your user name, and then press
  the Enter key.";
      }
      else if (validName(userName))
      {
        greeting = "Hello, " + userName + ".";
      }
      else
      {
        greeting = "Sorry, " + userName + ", you are not on the list.";
      }
      return greeting;
    }

    /**
     * Checks whether a name is in the validNames list.
     */
    public static function validName(inputName:String = ""):Boolean
    {
      if (validNames.indexOf(inputName) > -1)
      {
        return true;
      }
      else
      {
        return false;
      }
    }
  }
}
```

Greeter 클래스에 여러 가지 새로운 기능이 포함되었습니다.

- `validNames` 배열에 유효한 사용자 이름이 나열됩니다. 이 배열은 Greeter 클래스가 로드되면 세 개의 이름이 있는 목록으로 초기화됩니다.
- 이제 `sayHello()` 메서드에서 사용자 이름을 받아들이며 일부 조건에 따라 인사말을 변경합니다. `userName`이 비어 있는 문자열(`""`)인 경우 사용자에게 이름을 묻는 메시지가 나타나도록 `greeting` 속성이 설정됩니다. 사용자 이름이 유효한 경우 인사말은 `"Hello, userName"`입니다. 마지막으로 위의 두 조건 모두 충족되지 않은 경우 `greeting` 변수가 `"Sorry, userName, you are not on the list."`로 설정됩니다.

- validNames 배열에 inputName이 있으면 validName() 메서드가 true를 반환하고 그렇지 않으면 false를 반환합니다. validNames.indexOf(inputName) 명령문은 inputName 문자열에 대해 validNames 배열에 있는 각 문자열을 검사합니다. Array.indexOf() 메서드는 배열에 있는 객체의 첫 번째 인스턴스에 대한 인덱스 위치를 반환하고, 배열에 객체가 없는 경우에는 값 -1을 반환합니다.

다음으로 이 ActionScript 클래스를 참조하는 Flash 또는 Flex 파일을 편집합니다.

Flash 제작 도구를 사용하여 응용 프로그램을 수정하려면:

1. HelloWorld.fla 파일을 엽니다.
2. 비어 있는 문자열("")이 Greeter 클래스의 sayHello() 메서드에 전달되도록 프레임 1의 스크립트를 다음과 같이 수정합니다.

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");
```

3. [도구] 팔레트에서 [텍스트 도구]를 선택한 다음 스테이지에 새 텍스트 필드를 두 개 만듭니다. 이때 두 텍스트 필드를 기존의 mainText 텍스트 필드 바로 아래에 나란히 배치합니다.
4. 첫 번째 새 텍스트 필드에 레이블 역할을 할 텍스트 **User Name:**을 입력합니다.
5. 나머지 새 텍스트 필드를 선택한 다음 속성 관리자에서 텍스트 필드 유형으로 [InputText]를 선택합니다. 인스턴스 이름으로 textIn을 입력합니다.
6. 기본 타임라인의 첫 번째 프레임을 클릭합니다.
7. [액션] 패널에서 기존 스크립트의 맨 끝에 다음 행을 추가합니다.

```
mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
  if (event.keyCode == Keyboard.ENTER)
  {
    mainText.text = myGreeter.sayHello(textIn.text);
  }
}
```

이 새 코드는 다음 기능을 추가합니다.

- 처음 두 행은 두 텍스트 필드의 경계를 정의하는 역할만 합니다.
- textIn 필드와 같은 입력 텍스트 필드에는 전달 가능한 이벤트 집합이 포함되어 있습니다. addEventListener() 메서드를 사용하여 특정 유형의 이벤트 발생 시 실행되는 함수를 정의할 수 있습니다. 이 경우의 이벤트는 키보드의 Enter 키를 누르는 동작입니다.

- `keyPressed()` 사용자 정의 함수는 `myGreeter` 객체의 `sayHello()` 메서드를 호출하여 `textIn` 텍스트 필드의 텍스트를 매개 변수로 전달합니다. 이 메서드는 전달된 값을 기반으로 하여 인사말 문자열을 반환합니다. 그러면 반환된 문자열은 `mainText` 텍스트 필드의 `text` 속성에 할당됩니다.

프레임 1에 대한 전체 스크립트는 다음과 같습니다.

```
mainText.border = true;
textIn.border = true;

var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
  if (event.keyCode == Keyboard.ENTER)
  {
    mainText.text = myGreeter.sayHello(textIn.text);
  }
}
```

8. 파일을 저장합니다.
9. [컨트롤] > [무비 테스트]를 선택하여 응용 프로그램을 실행합니다.

   응용 프로그램을 실행하면 사용자 이름을 입력하라는 메시지가 나타납니다. 사용자 이름이 유효하면(예: Sammy, Frank, Dean 등) 응용 프로그램에서는 "hello"라는 확인 메시지를 표시합니다.

후속 예제 실행

"Hello World" ActionScript 3.0 응용 프로그램을 개발하고 실행해 보았으므로 이 설명서에 있는 다른 코드 예제를 실행하는 데 필요한 기본적인 지식을 익혔을 것입니다.

이 장에 제시된 예제 코드 샘플 테스트

이 설명서를 참고하여 작업하면서, 여러 항목을 설명하는 데 사용된 예제 코드 샘플을 테스트해 볼 수도 있습니다. 이러한 테스트는 프로그램에서 특정 지점의 변수 값을 표시하는 것일 수도 있고, 화면 내용을 보거나 상호 작용하는 것일 수도 있습니다. 시각적 내용이나 상호 작용 등을 테스트하는 경우, 코드 샘플 앞이나 코드 샘플 내에 필수 요소가 설명됩니다. 코드를 테스트하려면 이 설명된 요소가 포함된 문서를 작성하기만 하면 됩니다. 프로그램에서 특정 지점의 변수 값을 보려는 경우에는 몇 가지 방법이 있습니다. 한 가지 방법은 디버거(예: Flex Builder 및 Flash에 내장된 디버거)를 사용하는 것입니다. 하지만 단순한 테스트를 수행하는 경우에는 변수 값을 사용자가 볼 수 있는 대상으로 인쇄하는 것이 가장 쉽습니다.

다음 단계를 따라 Flash 문서를 만들어 코드 샘플을 테스트하고 변수 값을 볼 수 있습니다.

이 장에 제시된 예제를 테스트할 Flash 문서를 만들려면:

1. 새 Flash 문서를 만들어 하드 드라이브에 저장합니다.
2. 스테이지의 텍스트 필드에 테스트 값을 표시하려면 [텍스트 도구]를 활성화하고 스테이지에 새 동적 텍스트 필드를 만듭니다. 폭과 높이가 큰 텍스트 필드에 [행 유형]을 [여러 행]으로 설정하고 테두리를 적용하면 유용합니다. 속성 관리자에서 이 텍스트 필드의 인스턴트 이름을 지정합니다(예: "outputText"). 이 텍스트 필드에 값을 기록하려면 `appendText()` 메서드를 호출하는 코드를 예제 코드(아래 참조)에 추가합니다.
3. 또는 `trace()` 함수 호출을 코드 샘플(아래 참조)에 추가하여 예제의 결과를 볼 수도 있습니다.
4. 주어진 예제를 테스트하려면 [액션] 패널에 코드 샘플을 복사합니다. 필요한 경우에는 `trace()` 함수 호출을 추가하거나 `appendText()` 메서드를 사용하여 텍스트 필드에 값을 추가합니다.
5. 주 메뉴에서 [컨트롤] > [무비 테스트]를 선택하여 SWF 파일을 만들고 그 결과를 봅니다.

이러한 방법은 변수 값을 보려는 것으로, 예제를 테스트할 때 변수 값을 쉽게 볼 수 있는 방법은 두 가지가 있습니다. 즉, 스테이지의 텍스트 필드 인스턴스에 값을 기록하는 방법과 `trace()` 함수를 사용하여 값을 [출력] 패널로 출력하는 방법이 있습니다.

- `trace()` 함수: ActionScript `trace()` 함수는 이 함수로 전달되는 모든 매개 변수의 값(변수 또는 리터럴 표현식)을 [출력] 패널에 기록합니다. 이 설명서에 나오는 많은 예제 샘플에 `trace()` 함수 호출이 포함되어 있으므로, 이러한 샘플을 활용할 때 사용자는 문서에 코드를 복사해서 프로젝트를 테스트하기만 하면 됩니다. `trace()`가 포함되어 있지 않은 코드 샘플의 변수 값을 테스트하는 데 이 함수 호출을 사용하려면 `trace()` 호출을 코드 샘플에 추가하여 변수를 매개 변수로 전달하면 됩니다. 예를 들어, 해당 장에서 제공된 이러한 코드 샘플이 있는 경우 다음과 같이 합니다.

  ```
  var albumName:String = "Three for the money";
  ```

  [액션] 패널에 이 코드를 복사한 후, 다음과 같이 `trace()` 함수에 대한 호출을 추가하여 코드 샘플의 결과를 테스트합니다.

  ```
  var albumName:String = "Three for the money";
  trace("albumName =", albumName);
  ```

  프로그램을 실행하면 다음 행이 출력됩니다.

  ```
  albumName = Three for the money
  ```

  각 `trace()` 함수 호출에 여러 매개 변수가 전달되어 하나의 행으로 출력될 수 있습니다. 각 `trace()` 함수 호출 끝에 줄바꿈이 추가되어 개별 `trace()` 호출이 각각 다른 행에 출력됩니다.

- 스테이지의 텍스트 필드: `trace()` 호출을 사용하지 않을 경우, [텍스트 도구]를 사용하여 스테이지에 동적 텍스트 필드를 추가하여 이 텍스트 필드에 값을 기록한 후 코드 샘플 결과를 볼 수 있습니다. TextField 클래스의 `appendText()` 메서드를 사용하여 텍스트 필드의 내용 끝에 문자열 값을 추가할 수 있습니다. ActionScript를 사용하여 텍스트 필드에 액세스하려면 속성 관리자에서 이 텍스트 필드에 대한 인스턴스 이름을 지정해야 합니다. 예를 들어 텍스트 필드의 인스턴스 이름이 `outputText`인 경우, 다음 코드를 사용하여 변수 `albumName`의 값을 확인할 수 있습니다.

  ```
  var albumName:String = "Three for the money";
  outputText.appendText("albumName = ");
  outputText.appendText(albumName);
  ```

  이 코드는 `outputText`라는 텍스트 필드에 다음과 같은 텍스트를 기록합니다.

  ```
  albumName = Three for the money
  ```

  위의 예제에서 볼 수 있듯이 `appendText()` 메서드는 이전 내용과 동일한 행에 텍스트를 추가합니다. 따라서 여러 `appendText()` 호출을 사용하여 동일한 텍스트 행에 여러 값을 추가할 수 있습니다. 텍스트를 다음 행으로 배치하려면 개행 문자(`"\n"`)를 추가할 수 있습니다.

  ```
  outputText.appendText("\n"); // 텍스트 필드에 줄 바꿈을 추가합니다.
  ```

  `appendText()` 메서드는 `trace()` 함수와는 달리 하나의 값만을 매개 변수로 받으며, 이 값은 문자열이어야 합니다(문자열 인스턴스 또는 문자열 리터럴). 문자열이 아닌 변수를 출력하려면 먼저 값을 문자열로 변환해야 합니다. 가장 쉬운 변환 방법은 객체의 `toString()` 메서드를 호출하는 것입니다.

  ```
  var albumYear:int = 1999;
  outputText.appendText("albumYear = ");
  outputText.appendText(albumYear.toString());
  ```

이 장의 마무리 예제를 사용한 작업

이 장처럼 본 설명서의 대부분 장에는 해당 장에서 설명한 여러 개념에 관련된 중요한 마무리 예제가 포함되어 있습니다. 하지만 이 장의 Hello World 예제와는 달리 마무리 예제는 단계별 자습서 형식으로 표시되지 않습니다. 각 예제에 관련 ActionScript 3.0 코드가 강조 표시되어 설명되지만 특정 개발 환경에서 예제를 실행하는 방법에 대한 지침은 제공되지 않습니다. 그러나 이 설명서와 함께 배포된 예제 파일에는 필요한 파일이 모두 포함되어 있으므로 선택한 개발 환경에서 예제를 쉽게 컴파일하고 실행할 수 있습니다.

제 3 장

ActionScript 언어 및 구문

3

ActionScript 3.0은 기본 ActionScript 언어와 Adobe Flash Player API(Application Programming Interface)로 구성되어 있습니다. 기본 언어는 기본 ECMAScript(ECMA-262) Edition 4 초안 언어 사양을 구현하는 ActionScript의 일부입니다. Flash Player API는 프로그래밍 방식으로 Flash Player에 액세스할 때 사용합니다.

이 장에서는 기본 ActionScript 언어와 구문에 대해 간략히 소개합니다. 이 장을 학습하면 데이터 유형과 변수를 다루는 방법, 적합한 구문을 사용하는 방법 및 프로그램에서 데이터 흐름을 제어하는 방법에 대한 기본 개념을 이해할 수 있을 것입니다.

목차

언어 개요

객체는 ActionScript 3.0 언어의 핵심 개념이며 기본적인 구성 단위입니다. 객체는 선언된 모든 변수, 작성된 모든 함수 및 만들어진 모든 클래스 인스턴스입니다. ActionScript 3.0 프로그램은 작업을 수행하고 이벤트에 응답하고 서로 통신하는 객체의 그룹으로 간주할 수 있습니다.

Java 또는 C++의 OOP(객체 지향 프로그래밍)에 익숙한 프로그래머는 객체를 두 가지 멤버가 포함된 모듈로 생각할 수 있습니다. 즉, 데이터는 멤버 변수나 속성에 저장되고 비헤이비어는 메서드를 통해 액세스할 수 있는 것으로 생각할 수 있습니다. ActionScript 3.0의 기반이 되는 ECMAScript Edition 4 초안에서는 이와 유사하지만 조금 다른 방식으로 객체를 정의합니다. ECMAScript 초안에서 객체는 단지 속성 모음일 뿐입니다. 이러한 속성은 데이터뿐만 아니라 함수 또는 다른 객체도 저장할 수 있는 컨테이너입니다. 이 방식으로 객체에 연결되는 함수를 메서드라고 합니다.

Java 또는 C++에 대한 지식이 있는 프로그래머에게 ECMAScript 초안 정의가 조금 이상하게 보일 수도 있지만 실제로 ActionScript 3.0 클래스를 사용하여 객체 유형을 정의하는 것은 Java 또는 C++로 클래스를 정의하는 방식과 매우 유사합니다. ActionScript 객체 모델과 기타 고급 항목을 설명하는 경우에는 두 객체 정의를 구분하는 것이 중요하지만 대부분의 경우 *속성*이라는 용어는 메서드가 아닌 클래스 멤버 변수를 의미합니다. 예를 들어, *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 *속성*이라는 용어는 변수 또는 getter-setter 속성을 의미하며 *메서드*라는 용어는 클래스의 일부인 함수를 의미합니다.

Java 또는 C++와 달리 ActionScript에서는 클래스가 추상적인 엔터티만은 아니라는 미묘한 차이점이 있습니다. ActionScript 클래스는 클래스의 속성과 메서드를 저장하는 *클래스 객체*로 구체화됩니다. 이로 인해 클래스 또는 패키지의 최상위에서 명령문이나 실행 코드를 포함하는 등 Java 및 C++ 프로그래머에게는 생소할 수 있는 기법을 사용하는 것이 가능합니다.

ActionScript 클래스와 Java 또는 C++ 클래스 간의 또 다른 차이점은 모든 ActionScript 클래스에는 *프로토타입 객체*가 있다는 점입니다. 이전 버전의 ActionScript에서는 여러 프로토타입 객체를 *프로토타입 체인*에 함께 연결하여 전체 클래스 상속 계층 구조의 기초로 제공했습니다. 그러나 ActionScript 3.0의 상속 시스템에서는 프로토타입 객체가 작은 역할만 담당합니다. 비록 역할이 축소되기는 했지만 클래스의 모든 인스턴스에서 속성과 해당 값을 공유하려는 경우 이전처럼 프로토타입 객체를 정적 속성 및 메서드 대신 사용할 수 있습니다.

이전에는 고급 ActionScript 프로그래머가 특수 내장 언어 요소를 사용하여 프로토타입 체인을 직접 조작할 수 있었습니다. 언어 차원의 클래스 기반 프로그래밍 인터페이스가 더욱 강화되면서 `__proto__` 및 `__resolve` 등 대부분의 특수 언어 요소는 더 이상 사용할 수 없게 되었습니다. 또한 내부 상속 메커니즘을 최적화한 결과 Flash Player의 성능은 크게 향상되었지만 상속 메커니즘에 직접 액세스할 수 없게 되었습니다.

객체 및 클래스

ActionScript 3.0에서 모든 객체는 클래스로 정의됩니다. 클래스는 객체 유형의 청사진 또는 템플릿으로 간주할 수 있습니다. 클래스 정의에 변수, 상수 및 메서드를 포함할 수 있습니다. 변수 및 상수에는 데이터 값이 저장되며, 메서드는 클래스에 연결된 비헤이비어를 캡슐화하는 함수입니다. 속성에 저장된 값은 *프리미티브 값* 또는 다른 객체일 수 있습니다. 프리미티브 값에는 Number, String 및 부울 값이 있습니다.

ActionScript에는 여러 내장 클래스가 기본 언어의 일부로 포함되어 있습니다. Number, Boolean 및 String과 같은 일부 내장 클래스는 ActionScript에서 사용할 수 있는 프리미티브 값을 나타냅니다. Array, Math 및 XML과 같은 기타 클래스는 ECMAScript 표준의 일부인 보다 복잡한 객체를 정의합니다.

내장되어 있거나 사용자 정의된 모든 클래스는 Object 클래스에서 파생됩니다. 이전의 ActionScript에 익숙한 프로그래머의 경우, 모든 클래스가 여전히 Object 클래스에서 파생되지만 Object 데이터 유형이 더 이상 기본 데이터 유형이 아니라는 점에 유의해야 합니다. ActionScript 2.0에서는 유형 약어가 생략된 변수를 Object 유형으로 간주하므로 다음과 같은 두 개의 코드 행이 동일했습니다.

```
var someObj:Object;
var someObj;
```

그러나 ActionScript 3.0에서는 유형이 지정되지 않은 변수의 개념을 소개하며 이는 다음과 같은 두 가지 방법으로 지정할 수 있습니다.

```
var someObj:*;
var someObj;
```

유형이 지정되지 않은 변수는 Object 유형의 변수와 다릅니다. 이러한 변수 간의 주요 차이점은 유형이 지정되지 않은 변수에는 `undefined` 특수 값을 저장할 수 있는 반면 Object 유형의 변수에는 이 값을 저장할 수 없다는 점입니다.

`class` 키워드를 사용하여 클래스를 직접 정의할 수 있습니다. 클래스 속성은 세 가지 방법으로 선언할 수 있습니다. 상수는 `const` 키워드를 사용하여 정의하고 변수는 `var` 키워드를 사용하여 정의하며 getter 및 setter 속성은 메서드 선언에서 `get` 및 `set` 특성을 사용하여 정의할 수 있습니다. 메서드는 `function` 키워드를 사용하여 선언할 수 있습니다.

`new` 연산자를 사용하여 클래스의 인스턴스를 만듭니다. 다음 예제에서는 `myBirthday`라는 Date 클래스의 인스턴스를 만듭니다.

```
var myBirthday:Date = new Date();
```

패키지 및 네임스페이스

패키지와 네임스페이스는 서로 관련된 개념입니다. 패키지를 사용하면 클래스 정의를 묶음 처리하여 쉽게 코드를 공유하고 이름이 충돌하는 것을 최소화할 수 있습니다. 네임스페이스를 사용하면 속성 및 메서드 이름과 같은 식별자의 가시성을 제어할 수 있습니다. 네임스페이스는 패키지 내부 또는 외부에 있는 코드에 적용할 수 있습니다. 패키지를 사용하여 클래스 파일을 구성할 수 있으며, 네임스페이스를 사용하여 개별 속성 및 메서드의 가시성을 관리할 수 있습니다.

패키지

ActionScript 3.0의 패키지는 네임스페이스로 구현되지만 네임스페이스와 동일한 의미로 사용되지는 않습니다. 패키지를 선언하면 특수한 유형의 네임스페이스가 자동으로 만들어지며 이 네임스페이스는 컴파일 타임에 확인할 수 있습니다. 명시적으로 만든 네임스페이스는 컴파일 타임에 확인되지 않을 수도 있습니다.

다음 예제에서는 `package` 지시문을 사용하여 하나의 클래스가 들어 있는 단순한 패키지를 만듭니다.

```
package samples
{
  public class SampleCode
  {
    public var sampleGreeting:String;
    public function sampleFunction()
    {
      trace(sampleGreeting + " from sampleFunction()");
    }
  }
}
```

이 예제에서 클래스 이름은 SampleCode입니다. 이 클래스는 samples 패키지 내에 있기 때문에 SampleCode라는 클래스 이름은 컴파일 타임에 컴파일러에 의해 samples.SampleCode로 자동적으로 정규화됩니다. 또한 컴파일러에서 속성 또는 메서드 이름을 모두 정규화하므로 `sampleGreeting` 및 `sampleFunction()`은 각각 `samples.SampleCode.sampleGreeting` 및 `samples.SampleCode.sampleFunction()`으로 정규화됩니다.

대부분의 개발자 특히, Java 프로그래밍 경력이 있는 개발자는 패키지의 최상위에 클래스만 배치하려고 할 수 있습니다. 그러나 ActionScript 3.0에서는 패키지의 최상위에 클래스뿐만 아니라 변수, 함수 및 명령문도 배치할 수 있습니다. 이 기능의 고급 용도 중 하나는 패키지의 최상위에 네임스페이스를 정의하여 해당 패키지의 모든 클래스에서 사용할 수 있다는 것입니다. 그러나 패키지의 최상위에서는 두 개의 액세스 지정자 즉, `public` 및 `internal`만 사용할 수 있습니다. 중첩된 클래스를 전용으로 선언할 수 있는 Java와 달리 ActionScript 3.0에서는 중첩된 클래스와 전용 클래스 어느 쪽도 지원하지 않습니다.

ActionScript 3.0 패키지는 여러 면에서 Java 프로그래밍 언어로 구현된 패키지와 유사합니다. 이전 예제에서 확인할 수 있듯이 Java의 경우와 마찬가지로 도트 연산자(.)를 사용하여 정규화된 패키지 참조를 표시합니다. 패키지를 사용하면 코드를 직관적인 계층 구조로 구성하여 다른 프로그래머에게 제공할 수 있습니다. 이와 같이 다른 사람과 공유할 패키지를 만들고 다른 사람이 만든 패키지를 코드에 사용할 수 있으므로 코드 공유가 촉진됩니다.

패키지를 사용하면 사용할 식별자 이름을 고유하게 만들어 다른 식별자 이름과 충돌하지 않도록 할 수도 있습니다. 실제로 일부 사용자는 이 기능이 패키지의 주요 장점이라고 말합니다. 예를 들어, 서로의 코드를 공유하려는 두 명의 프로그래머가 SampleCode라는 클래스를 각각 만들 수 있습니다. 패키지를 사용하지 않으면 이름이 충돌하며 이 문제는 하나의 클래스 이름을 변경해야 해결할 수 있습니다. 그러나 패키지를 사용하면 두 클래스 중 하나를 패키지에 배치하거나 둘 모두를 이름이 다른 두 개의 패키지에 각각 배치하여 이름 충돌을 피할 수 있습니다.

패키지 이름에 도트를 사용하여 중첩된 패키지를 만들 수도 있습니다. 이렇게 하면 패키지의 계층적 구조를 만들 수 있습니다. Flash Player API에서 제공하는 flash.xml 패키지는 이에 대한 좋은 예입니다. flash.xml 패키지는 flash 패키지 내에 중첩됩니다.

flash.xml 패키지에는 이전 버전의 ActionScript에서 사용된 기존 XML 파서가 포함되어 있습니다. 현재 flash.xml 패키지에 기존 XML 파서가 있는 이유 중 하나는 ActionScript 3.0에서 사용할 수 있는 E4X(ECMAScript for XML) 사양 기능을 구현하는 새 XML 클래스 이름과 이전 XML 클래스 이름이 충돌한다는 것입니다.

이전 XML 클래스를 패키지로 이동한 것은 일단 좋은 방법이기는 하지만 대부분의 이전 XML 클래스 사용자가 flash.xml 패키지를 가져올 것으로 보이므로 이전 XML 클래스의 정규화된 이름(즉, flash.xml.XML)을 사용하지 않아 위와 동일한 이름 충돌이 발생할 가능성이 있습니다. 이를 피하기 위해 다음 예제에 나와 있듯이 이전 XML 클래스의 이름을 XMLDocument로 바꾸었습니다.

```
package flash.xml
{
  class XMLDocument {}
  class XMLNode {}
  class XMLSocket {}
}
```

Flash Player API는 대부분 flash 패키지 아래에 구성됩니다. 예를 들어, flash.display 패키지에 표시 목록 API가 포함되어 있으며, flash.events 패키지에는 새 이벤트 모델이 포함되어 있습니다.

패키지 만들기

ActionScript 3.0에서는 패키지, 클래스 및 소스 파일을 매우 유연하게 구성할 수 있습니다. 이전 버전의 ActionScript에서는 소스 파일당 하나의 클래스만 허용되었으며 소스 파일 이름이 클래스 이름과 일치해야 했습니다. ActionScript 3.0에서는 하나의 소스 파일에 여러 클래스를 포함할 수 있지만 해당 파일의 외부에 있는 코드에는 각 파일의 클래스를 하나만 사용할 수 있습니다. 즉, 패키지 선언 내에는 각 파일의 클래스를 하나만 선언할 수 있습니다. 추가적인 클래스는 패키지 정의 외부에 선언하여 소스 파일 외부에 있는 코드에서 추가적인 클래스를 참조할 수 없도록 합니다. 패키지 정의 내에 선언된 클래스 이름은 소스 파일 이름과 일치해야 합니다.

ActionScript 3.0에서 패키지를 보다 융통성 있게 선언할 수 있습니다. 이전 버전의 ActionScript에서 패키지는 단순히 소스 파일이 있는 디렉토리를 나타냈으며 사용자는 `package` 문을 사용하여 패키지를 선언하지 않고 클래스 선언에 정규화된 클래스 이름의 일부로 패키지 이름을 포함했습니다. ActionScript 3.0에서도 패키지는 디렉토리를 나타내지만 패키지에 클래스 이외의 요소를 포함할 수 있습니다. ActionScript 3.0에서는 `package` 문을 사용하여 패키지를 선언하므로 패키지의 최상위에 변수, 함수 및 네임스페이스도 선언할 수 있습니다. 또한 패키지의 최상위에 실행문을 포함할 수 있습니다. 패키지의 최상위에 변수, 함수 또는 네임스페이스를 선언한 경우 해당 수준에서 `public` 및 `internal` 특성만 사용할 수 있으며, 선언 대상이 클래스, 변수, 함수 또는 네임스페이스인지에 관계없이 파일당 하나의 패키지 수준 선언에서만 `public` 특성을 사용할 수 있습니다.

패키지는 코드를 체계화하고 이름 충돌을 방지하는 데 유용합니다. 패키지 개념은 클래스 상속 개념과 관련이 없으므로 이 둘을 혼동해서는 안됩니다. 동일한 패키지에 있는 두 개의 클래스는 공통된 네임스페이스에 있지만 그 외에 다른 방식으로 상호 관련되어야 하는 것은 아닙니다. 마찬가지로 중첩된 패키지와 상위 패키지 간에 아무런 의미적 관련성이 없을 수 있습니다.

패키지 가져오기

패키지 내의 클래스를 사용하려면 패키지 또는 특정 클래스 중 하나를 가져와야 합니다. 이는 클래스 가져오기를 생략해도 되었던 ActionScript 2.0과 다른 점입니다.

예를 들어, 이 장의 앞 부분에서 설명한 SampleCode 클래스 예제에 대해 살펴 보십시오. samples라는 패키지에 있는 SampleCode 클래스를 사용하려면 다음 import 문 중 하나를 사용해야 합니다.

```
import samples.*;
```

또는

```
import samples.SampleCode;
```

일반적으로 import 문은 최대한 명확해야 합니다. samples 패키지의 SampleCode 클래스만 사용하려는 경우 SampleCode 클래스가 속해 있는 전체 패키지가 아닌 SampleCode 클래스만 가져와야 합니다. 전체 패키지를 가져오면 예기치 못한 이름 충돌이 발생할 수 있습니다.

패키지 또는 클래스가 정의된 소스 코드 역시 *클래스 경로* 내에 배치해야 합니다. 패키지 경로는 컴파일러에게 가져올 패키지 및 클래스의 검색 위치를 알려 주기 위해 로컬 디렉토리 경로를 사용자 정의한 목록입니다. 클래스 경로는 *빌드 경로* 또는 *소스 경로*라고도 합니다.

클래스 또는 패키지를 올바르게 가져오면 정규화된 클래스 이름(samples.SampleCode) 또는 클래스 자체 이름(SampleCode) 중 하나를 사용할 수 있습니다.

정규화된 이름은 동일하게 이름이 지정된 클래스, 메서드 또는 속성으로 인해 코드가 모호해지는 경우 유용하지만 모든 식별자에 사용하면 관리하기 어려울 수 있습니다. 예를 들어, SampleCode 클래스 인스턴스를 인스턴스화할 때 정규화된 이름을 사용하면 코드가 길어집니다.

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

패키지가 중첩될수록 코드의 가독성은 떨어집니다. 모호한 식별자가 문제가 되지 않는 경우에는 간단한 식별자를 사용하여 더 읽기 쉬운 코드를 만들 수 있습니다. 예를 들어, 클래스 식별자만 사용하는 경우 다음과 같이 SampleCode 클래스의 새 인스턴스를 인스턴스화하는 것이 훨씬 간단합니다.

```
var mySample:SampleCode = new SampleCode();
```

먼저 적절한 패키지 또는 클래스를 가져오지 않고 식별자 이름을 사용하려고 하면 컴파일러에서 클래스 정의를 찾을 수 없게 됩니다. 반면에 패키지 또는 클래스를 가져온 경우 가져온 이름과 충돌하는 이름을 정의하려고 하면 오류가 발생합니다.

패키지를 만든 경우 해당 패키지의 모든 멤버에 대한 기본 액세스 지정자는 internal입니다. 이는 기본적으로 패키지 멤버만 해당 패키지의 다른 멤버를 참조할 수 있다는 것을 의미합니다. 패키지 외부의 코드에 클래스를 사용하려면 클래스를 public으로 선언해야 합니다.
예를 들어, 다음 패키지에 SampleCode 및 CodeFormatter 클래스가 포함되어 있습니다.

```
// SampleCode.as 파일
package samples
{
  public class SampleCode {}
}

// CodeFormatter.as 파일
package samples
{
  class CodeFormatter {}
}
```

SampleCode 클래스는 public 클래스로 선언되어 있으므로 패키지 외부에서 참조할 수 있습니다. 그러나 CodeFormatter 클래스는 samples 패키지 내에서만 참조할 수 있습니다. samples 패키지 외부에서 CodeFormatter 클래스에 액세스하려고 하면 다음 예제와 같이 오류가 발생합니다.

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // public 클래스
var myFormatter:CodeFormatter = new CodeFormatter(); // 오류
```

패키지 외부에서 두 클래스를 모두 사용할 수 있게 하려면 두 클래스 모두 public으로 선언해야 합니다. 패키지 선언에는 public 특성을 적용할 수 없습니다.

정규화된 이름은 패키지를 사용할 때 발생할 수 있는 이름 충돌 문제를 해결하는 데 유용합니다. 동일한 식별자를 가진 클래스가 정의된 두 개의 패키지를 가져오는 경우 이와 같은 문제가 발생할 수 있습니다. 예를 들어, SampleCode라는 클래스가 포함되어 있는 또 다른 패키지를 살펴 보십시오.

```
package langref.samples
{
  public class SampleCode {}
}
```

두 클래스를 모두 가져오면 SampleCode 클래스를 참조할 때 다음과 같이 이름이 충돌하게 됩니다.

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // 이름 충돌
```

컴파일러는 어떤 SampleCode 클래스를 사용해야 하는지 알 수 없습니다. 이 충돌 문제를 해결하려면 다음과 같이 각 클래스의 정규화된 이름을 사용해야 합니다.

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

참고

C++ 사용 경력이 있는 프로그래머가 import 문과 #include 문을 혼동하는 경우가 많습니다. C++ 컴파일러의 경우 파일을 한 번에 하나씩만 처리하고 헤더 파일이 명시적으로 포함되어 있지 않으면 다른 파일에서 클래스 정의를 찾지 않으므로 C++에서 #include 지시문은 필수적인 요소입니다. ActionScript 3.0에도 include 지시문이 포함되어 있지만 클래스 및 패키지를 가져오는 데 사용되지 않습니다. ActionScript 3.0에서 클래스 또는 패키지를 가져오려면 import 문을 사용해야 하며, 패키지가 포함된 소스 파일을 클래스 경로에 배치해야 합니다.

네임스페이스

네임스페이스를 사용하면 사용자가 만든 속성 및 메서드의 가시성을 제어할 수 있습니다. public, private, protected 및 internal 액세스 제어 지정자를 내장 네임스페이스로 간주할 수 있습니다. 이러한 미리 정의된 액세스 제어 지정자가 사용자 요구에 적합하지 않은 경우 직접 네임스페이스를 만들 수 있습니다.

ActionScript 구현의 구문 및 세부 사항이 XML의 구문 및 세부 사항과 약간 다르지만 XML 네임스페이스에 익숙한 사용자에게는 이 설명서의 내용이 낯설게 느껴지지 않을 것입니다. 네임스페이스의 개념은 매우 간단하지만 이전에 네임스페이스를 사용하여 작업한 경험이 없는 경우 이를 구현하려면 특정 용어를 익혀야 합니다.

네임스페이스의 작동을 이해하려면 속성 또는 메서드 이름에 식별자 및 네임스페이스가 항상 포함된다는 점을 이해하는 것이 좋습니다. 식별자는 일반적으로 이름으로 간주되는 것을 말합니다. 예를 들어, 다음 클래스 정의에서 식별자는 sampleGreeting과 sampleFunction()입니다.

```
class SampleCode
{
  var sampleGreeting:String;
  function sampleFunction () {
    trace(sampleGreeting + " from sampleFunction()");
  }
}
```

클래스 등을 정의할 때 네임스페이스 특성을 먼저 지정하지 않으면 해당 정의에 사용된 이름이 internal 네임스페이스로 정규화되므로 동일한 패키지에 있는 호출자만 해당 이름을 참조할 수 있습니다. Strict 모드로 설정된 컴파일러의 경우 네임스페이스 특성이 없는 모든 식별자에 internal 네임스페이스가 적용된다는 경고를 표시합니다. 식별자를 모든 범위에서 사용할 수 있도록 하려면 식별자 이름 앞에 public 특성을 명시해야 합니다. 이전 예제 코드에서 sampleGreeting 및 sampleFunction()에는 모두 internal 네임스페이스가 적용됩니다.

네임스페이스를 사용할 때 수행해야 할 세 가지 기본 단계는 다음과 같습니다. 첫 번째, namespace 키워드를 사용하여 네임스페이스를 정의해야 합니다. 예를 들어, 다음 코드에서는 version1 네임스페이스를 정의합니다.

```
namespace version1;
```

두 번째, 속성 또는 메서드 선언에서 액세스 제어 지정자 대신 네임스페이스를 사용하여 네임스페이스를 적용합니다. 다음 예제에서는 myFunction() 함수를 version1 네임스페이스에 배치합니다.

```
version1 function myFunction() {}
```

세 번째, 네임스페이스가 적용되면 use 지시문을 사용하거나 식별자 이름을 네임스페이스로 정규화하여 참조할 수 있습니다. 다음 예제에서는 use 지시문을 통해 myFunction() 함수를 참조합니다.

```
use namespace version1;
myFunction();
```

다음 예제에서와 같이 정규화된 이름을 사용하여 myFunction() 함수를 참조할 수도 있습니다.

```
version1::myFunction();
```

네임스페이스 정의

네임스페이스에는 URI(Uniform Resource Identifier)라는 값이 포함되며 이를 *네임스페이스 이름*이라고도 합니다. URI를 사용하여 네임스페이스 정의를 고유하게 만들 수 있습니다.

두 가지 방법 중 하나로 네임스페이스 정의를 선언하여 네임스페이스를 만듭니다. XML 네임스페이스를 정의하듯이 명시적 URI가 있는 네임스페이스를 정의할 수 있으나 URI를 생략하는 것도 가능합니다. 다음 예제에서는 URI를 사용하여 네임스페이스를 정의하는 방법을 보여 줍니다.

```
namespace flash_proxy = “http://www.adobe.com/flash/proxy”;
```

URI는 해당 네임스페이스에 대한 고유한 식별 문자열 역할을 합니다. URI를 생략하면 다음 예제에서와 같이 컴파일러에서 URI 대신 고유한 내부 식별 문자열을 만듭니다. 사용자는 이 내부 식별 문자열에 액세스하지 못합니다.

```
namespace flash_proxy;
```

URI 사용 여부와 관계없이 한 번 정의한 네임스페이스는 동일한 범위에서 다시 정의할 수 없습니다. 이전에 정의한 네임스페이스를 동일한 범위에서 정의하려고 하면 컴파일러 오류가 발생합니다.

패키지 또는 클래스 내에 네임스페이스가 정의된 경우 적절한 액세스 제어 지정자를 사용하지 않으면 해당 패키지 또는 클래스 외부의 코드에서 네임스페이스를 참조할 수 없습니다. 예를 들어, 다음 코드에서는 flash.utils 패키지 내에 정의된 flash_proxy 네임스페이스를 보여 줍니다. 다음 예제에서는 액세스 제어 지정자가 사용되지 않으므로 flash.utils 패키지 내의 코드에서만 flash_proxy 네임스페이스를 참조할 수 있고, 패키지 외부의 모든 코드에서는 flash_proxy 네임스페이스를 참조할 수 없습니다.

```
package flash.utils
{
  namespace flash_proxy;
}
```

다음 코드에서는 패키지 외부의 코드에서 flash_proxy 네임스페이스를 참조할 수 있도록 public 특성을 사용합니다.

```
package flash.utils
{
  public namespace flash_proxy;
}
```

네임스페이스 적용

네임스페이스를 적용한다는 것은 정의를 네임스페이스에 배치하는 것을 의미합니다. 네임스페이스에 배치할 수 있는 정의에는 함수, 변수 및 상수가 있으며 사용자 정의 네임스페이스에 클래스를 배치할 수는 없습니다.

예를 들어, public 액세스 제어 네임스페이스를 사용하여 선언된 함수를 살펴 보십시오. 함수 정의에서 public 특성을 사용하여 함수를 공용 네임스페이스에 배치하면 모든 코드에서 해당 함수를 사용할 수 있습니다. 네임스페이스를 정의하면 public 특성 사용법과 동일한 방법으로 정의된 네임스페이스를 사용할 수 있으며, 사용자 정의 네임스페이스를 참조할 수 있는 코드에서 해당 정의를 사용할 수 있습니다. 예를 들어, example1 네임스페이스를 정의하면 다음 예제에서와 같이 example1을 특성으로 사용하여 myFunction() 메서드를 추가할 수 있습니다.

```
namespace example1;
class someClass
{
  example1 myFunction() {}
}
```

example1 네임스페이스를 특성으로 사용하여 myFunction() 메서드를 선언하면 이 메서드가 example1 네임스페이스에 속하게 됩니다.

네임스페이스를 적용할 때에는 다음 사항에 주의해야 합니다.

- 각 선언에는 네임스페이스를 하나만 적용할 수 있습니다.
- 한 번에 둘 이상의 정의에 네임스페이스 특성을 적용할 수 없습니다. 즉, 서로 다른 10개의 함수에 네임스페이스를 적용하려면 10개의 각 함수 정의에 네임스페이스를 특성으로 추가해야 합니다.
- 또한 네임스페이스와 액세스 제어 지정자는 상호 배타적이므로 네임스페이스를 적용하는 경우 액세스 제어 지정자를 지정할 수 없습니다. 즉, 네임스페이스를 적용한 함수 또는 속성을 public, private, protected 또는 internal로 선언할 수 없습니다.

네임스페이스 참조

public, private, protected 및 internal 등의 액세스 제어 네임스페이스로 선언된 메서드 또는 속성을 사용할 때 네임스페이스를 명시적으로 참조할 필요는 없습니다. 이는 이러한 특수 네임스페이스에 대한 액세스는 컨텍스트에 의해 제어되기 때문입니다. 예를 들어, 정의를 private 네임스페이스에 배치하면 자동으로 동일한 클래스 내의 코드에서 사용할 수 있게 됩니다. 그러나 사용자가 정의한 네임스페이스에는 이와 같은 컨텍스트 민감도가 없습니다. 사용자 정의 네임스페이스에 배치한 메서드 또는 속성을 사용하려면 네임스페이스를 참조해야 합니다.

use namespace 지시문을 사용하여 네임스페이스를 참조하거나 이름 한정 기호(::)를 사용하여 네임스페이스로 이름을 정규화할 수 있습니다. use namespace 지시문을 사용하여 네임스페이스를 참조하면 네임스페이스가 "열려" 정규화되지 않은 모든 식별자에 해당 네임스페이스를 적용할 수 있습니다. 예를 들어, example1 네임스페이스를 정의한 경우 use namespace example1을 사용하여 해당 네임스페이스에 있는 이름에 액세스할 수 있습니다.

```
use namespace example1;
myFunction();
```

한 번에 둘 이상의 네임스페이스를 열 수 있습니다. use namespace를 사용하여 네임스페이스를 열면 네임스페이스가 열린 코드 블록 전체에서 열린 상태로 남아 있게 됩니다. 열린 네임스페이스를 명시적으로 닫을 수는 없습니다.

그러나 둘 이상의 네임스페이스가 열려 있으면 이름이 충돌할 가능성이 커집니다. 네임스페이스를 열지 않으려면 use namespace 지시문을 사용하는 대신 네임스페이스 및 이름 한정 기호로 메서드 또는 속성을 정규화하면 됩니다. 예를 들어, 다음 코드에서는 example1 네임스페이스를 사용하여 myFunction()이라는 이름을 정규화하는 방법을 보여 줍니다.

```
example1::myFunction();
```

네임스페이스 사용

Flash Player API의 일부인 flash.utils.Proxy 클래스에서 이름 충돌을 방지하는 데 사용되는 네임스페이스에 대한 실제 예를 찾아볼 수 있습니다. ActionScript 2.0의 Object.__resolve 속성 대신 Proxy 클래스를 사용하면 정의되지 않은 속성 또는 메서드에 대한 참조를 차단하여 오류가 발생하는 것을 막을 수 있습니다. 이름이 충돌하지 않도록 하려면 Proxy 클래스의 메서드를 모두 flash_proxy 네임스페이스에 배치합니다.

flash_proxy 네임스페이스가 사용되는 방법을 보다 잘 이해하려면 Proxy 클래스를 사용하는 방법을 알아야 합니다. Proxy 클래스의 기능은 이 클래스에서 상속된 클래스에서만 사용할 수 있습니다. 즉, 객체에 Proxy 클래스의 메서드를 사용하려면 객체의 클래스 정의에서 Proxy 클래스를 확장해야 합니다. 예를 들어, 정의되지 않은 메서드 호출을 차단하려면 Proxy 클래스를 확장한 다음 Proxy 클래스의 callProperty() 메서드를 재정의합니다.

일반적으로 네임스페이스 구현 과정이 네임스페이스 정의, 적용 및 참조의 3단계로 구성됨을 기억하고 있을 것입니다. 그러나 Proxy 클래스의 어떤 메서드도 명시적으로 호출되지 않기 때문에 flash_proxy 네임스페이스는 정의 및 적용되기는 하지만 참조되지는 않습니다. Flash Player API에서 flash_proxy 네임스페이스를 정의하고 이를 Proxy 클래스에 적용합니다. 코드에서는 Proxy 클래스를 확장하는 클래스에 flash_proxy 네임스페이스를 적용하기만 하면 됩니다.

다음과 유사한 방식으로 flash_proxy 네임스페이스가 flash.utils 패키지에 정의되어 있습니다.

```
package flash.utils
{
  public namespace flash_proxy;
}
```

이 네임스페이스는 Proxy 클래스에서 인용한 다음 예제와 같이 Proxy 클래스의 메서드에 적용됩니다.

```
public class Proxy
{
  flash_proxy function callProperty(name:*, ... rest):*
  flash_proxy function deleteProperty(name:*):Boolean
  ...
}
```

다음 코드에서 보듯이 먼저 Proxy 클래스와 flash_proxy 네임스페이스를 모두 가져와야 합니다. 그런 다음 Proxy 클래스를 확장하는 클래스를 선언해야 하며, Strict 모드에서 컴파일하는 경우에는 dynamic 특성도 추가해야 합니다. callProperty() 메서드를 재정의하는 경우 flash_proxy 네임스페이스를 사용해야 합니다.

```
package
{
  import flash.utils.Proxy;
  import flash.utils.flash_proxy;

  dynamic class MyProxy extends Proxy
  {
    flash_proxy override function callProperty(name:*, ...rest):*
    {
      trace("method call intercepted: " + name);
    }
  }
}
```

MyProxy 클래스의 인스턴스를 만들고 다음 예제에서 호출된 testing() 메서드처럼 정의되지 않은 메서드를 호출하는 경우, Proxy 객체에서 메서드 호출을 차단하고 재정의된 callProperty() 메서드 내의 명령문(이 경우 간단한 trace() 문)을 실행합니다.

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // 다음 메서드 호출 차단됨 : testing
```

flash_proxy 네임스페이스 내에 Proxy 클래스의 메서드를 배치하면 두 가지 장점이 있습니다. 첫 번째, 별도의 네임스페이스를 사용하면 Proxy 클래스를 확장하는 모든 클래스의 공용 인터페이스가 단순화됩니다. Proxy 클래스에서 재정의할 수 있는 메서드는 약 12개이며 모두 직접 호출되도록 설계되지 않았습니다. 공용 네임스페이스에 메서드를 모두 배치하면 혼동될 수 있습니다. 두 번째, Proxy의 하위 클래스에 Proxy 클래스의 메서드와 이름이 일치하는 인스턴스 메서드가 포함되어 있는 경우 flash_proxy 네임스페이스를 사용하면 이름 충돌이 방지됩니다. 예를 들어, 사용자가 정의한 메서드에 callProperty()라는 이름을 붙일 수 있습니다. 다음 코드에서는 사용자가 정의한 callProperty() 메서드가 다른 네임스페이스에 있으므로 오류가 발생하지 않습니다.

```
dynamic class MyProxy extends Proxy
{
  public function callProperty() {}
  flash_proxy override function callProperty(name:*, ...rest):*
  {
    trace("method call intercepted: " + name);
  }
}
```

네임스페이스는 public, private, internal 및 protected와 같은 네 가지 액세스 제어 지정자로는 불가능한 방식으로 메서드 또는 속성에 대한 액세스를 제공하려는 경우에도 유용합니다. 예를 들어, 몇 가지 유틸리티 메서드가 여러 패키지에 분산되어 있는 경우가 있을 수 있습니다. 이러한 메서드를 모든 패키지에서 사용할 수 있도록 하되 공용으로 지정하지 않으려고 합니다. 이러한 목적을 달성하려면 새 네임스페이스를 만들어 사용자 고유의 특수 액세스 제어 지정자로 사용하면 됩니다.

다음 예제에서는 사용자 정의 네임스페이스를 사용하여 다른 패키지에 있는 두 개의 함수를 그룹화합니다. 동일한 네임스페이스로 함수를 그룹화하면 클래스 또는 패키지에서 use namespace 문을 한 번만 사용하여 두 함수를 참조하도록 할 수 있습니다.

이 예제에서는 네 개의 파일을 사용하여 이러한 기법을 설명합니다. 네 개의 파일 모두 클래스 경로에 있어야 합니다. 첫 번째 파일인 myInternal.as는 myInternal 네임스페이스를 정의하는 데 사용됩니다. 파일이 example 패키지에 있으므로 해당 파일을 example 폴더에 배치해야 합니다. 네임스페이스가 public으로 표시되어 있으므로 다른 패키지로 가져올 수 있습니다.

```
// 폴더의 myInternal.as
package example
{
  public namespace myInternal = "http://www.adobe.com/2006/actionscript/
  examples";
}
```

두 번째와 세 번째 파일인 Utility.as 및 Helper.as에서는 다른 패키지에서도 사용할 수 있어야 하는 메서드가 포함된 클래스를 정의합니다. Utility 클래스가 example.alpha 패키지에 있으므로 example 폴더의 하위 폴더인 alpha 폴더에 파일을 배치해야 합니다. Helper 클래스가 example.beta 패키지에 있으므로 example 폴더의 하위 폴더인 beta 폴더에 파일을 배치해야 합니다. 이러한 example.alpha 및 example.beta 패키지를 사용하려면 네임스페이스를 가져와야 합니다.

```
// example/alpha 폴더의 Utility.as
package example.alpha
{
  import example.myInternal;

  public class Utility
  {
    private static var _taskCounter:int = 0;

    public static function someTask()
    {
      _taskCounter++;
    }

    myInternal static function get taskCounter():int
    {
      return _taskCounter;
    }
  }
}

// example/beta 폴더의 Helper.as
package example.beta
{
  import example.myInternal;

  public class Helper
  {
    private static var _timeStamp:Date;

    public static function someTask()
    {
      _timeStamp = new Date();
    }

    myInternal static function get lastCalled():Date
    {
      return _timeStamp;
    }
  }
}
```

네 번째 파일인 NamespaceUseCase.as에는 기본 응용 프로그램 클래스가 정의되어 있으며 example 폴더와 상위 폴더가 같은 폴더에 있어야 합니다. Adobe Flash CS3 Professional에서 이 클래스는 FLA의 문서 클래스로 사용됩니다. NamespaceUseCase 클래스는 myInternal 네임스페이스를 가져와서 이를 사용하여 다른 패키지에 들어 있는 두 개의 정적 메서드를 호출하기도 합니다. 예제에서는 코드를 단순화하기 위해서만 정적 메서드를 사용합니다. 정적 메서드와 인스턴스 메서드를 모두 myInternal 네임스페이스에 배치할 수 있습니다.

```
// NamespaceUseCase.as
package
{
  import flash.display.MovieClip;
  import example.myInternal;       // 네임스페이스를 가져옵니다 .
  import example.alpha.Utility;    // Utility 클래스를 가져옵니다 .
  import example.beta.Helper;      // Helper 클래스를 가져옵니다 .

  public class NamespaceUseCase extends MovieClip
  {
    public function NamespaceUseCase()
    {
      use namespace myInternal;

      Utility.someTask();
      Utility.someTask();
      trace(Utility.taskCounter); // 2

      Helper.someTask();
      trace(Helper.lastCalled);   // [someTask() 가 마지막으로 호출된 시간 ]
    }
  }
}
```

변수

변수를 사용하면 프로그램에서 사용하는 값을 저장할 수 있습니다. 변수를 선언하려면 변수 이름과 함께 var 문을 사용해야 합니다. ActionScript 2.0에서는 유형 약어를 사용하는 경우에만 var 문을 사용했습니다. ActionScript 3.0에서는 항상 var 문을 사용해야 합니다. 예를 들어 다음 ActionScript 행에서는 i라는 이름의 변수를 선언합니다.

```
var i;
```

변수를 선언할 때 var 문을 생략하면 Strict 모드에서는 컴파일러 오류가 발생하고 Standard 모드에서는 런타임 오류가 발생합니다. 예를 들어, i 변수가 이미 정의되어 있지 않으면 다음 코드 행에서 오류가 발생합니다.

```
i; // i 가 이전에 정의되지 않은 경우 오류가 발생합니다 .
```

변수를 데이터 유형과 연결하려면 변수를 선언할 때 연결해야 합니다. 변수 유형을 지정하지 않고 변수를 선언할 수 있지만 이럴 경우 Strict 모드에서는 컴파일러 경고가 발생합니다. 변수 이름 뒤에 콜론(:)과 변수 유형을 차례로 추가하여 변수 유형을 지정합니다. 예를 들어 다음 코드에서는 int 유형의 i 변수를 선언합니다.

```
var i:int;
```

대입 연산자(=)를 사용하여 변수에 값을 지정할 수 있습니다. 예를 들어, 다음 코드에서는 i 변수를 선언하고 값을 20으로 지정합니다.

```
var i:int;
i = 20;
```

다음 예제와 같이 변수를 선언하는 동시에 변수에 값을 지정하는 것이 보다 간편할 수 있습니다.

```
var i:int = 20;
```

변수를 선언할 때 변수에 값을 지정하는 방법은 정수 및 문자열과 같은 프리미티브 값을 지정하는 경우뿐만 아니라 배열을 만들거나 클래스의 인스턴스를 인스턴스화하는 경우에도 일반적으로 사용됩니다. 다음 예제에서는 하나의 코드 행으로 배열을 선언하고 값을 지정하는 방법을 보여 줍니다.

```
var numArray:Array = ["zero", "one", "two"];
```

new 연산자를 사용하여 클래스의 인스턴스를 만들 수 있습니다. 다음 예제에서는 CustomClass 인스턴스를 만들고 customItem 변수에 새로 만든 클래스 인스턴스에 대한 참조를 지정합니다.

```
var customItem:CustomClass = new CustomClass();
```

선언할 변수가 둘 이상이면 변수를 구분하는 쉼표 연산자(,)를 사용하여 하나의 코드 행에 모든 변수를 선언할 수 있습니다. 예를 들어, 다음 코드에서는 하나의 코드 행에 세 개의 변수를 선언합니다.

```
var a:int, b:int, c:int;
```

동일한 코드 행의 각 변수에 값을 지정할 수도 있습니다. 예를 들어, 다음 코드에서는 a, b 및 c 등의 세 개의 변수를 선언하고 각각 값을 지정합니다.

```
var a:int = 10, b:int = 20, c:int = 30;
```

쉼표 연산자를 사용하여 변수 선언을 명령문 하나로 그룹화할 수는 있지만 코드에 대한 가독성이 떨어질 수 있습니다.

변수 범위 이해

변수의 *범위*는 코드에서 어휘 참조로 변수에 액세스할 수 있는 영역입니다. *전역* 변수가 코드의 모든 영역에 정의되는 반면 *로컬* 변수는 코드의 한 부분에만 정의됩니다. ActionScript 3.0에서 변수는 변수가 선언되는 함수 또는 클래스의 범위가 항상 지정됩니다. 전역 변수는 모든 함수 또는 클래스 정의 외부에 정의된 변수입니다. 예를 들어, 다음 코드에서는 모든 함수 외부에 `strGlobal` 변수를 선언하여 전역 변수로 만듭니다. 이 예제에서는 함수 정의 내부와 외부에서 모두 사용할 수 있는 전역 변수를 보여 줍니다.

```
var strGlobal:String = "Global";
function scopeTest()
{
  trace(strGlobal); // 전역
}
scopeTest();
trace(strGlobal); // 전역
```

함수 정의 내부에 변수를 선언하여 로컬 변수를 선언합니다. 함수 정의는 로컬 변수를 정의할 수 있는 가장 작은 코드 영역입니다. 함수 내에 선언된 로컬 변수는 해당 함수 안에서만 존재합니다. 예를 들어, `localScope()` 함수 내에 `str2` 변수를 선언하는 경우 해당 변수를 함수 외부에서 사용할 수 없습니다.

```
function localScope()
{
  var strLocal:String = "local";
}
localScope();
trace(strLocal); // strLocal 이 전역적으로 정의되어 있지 않기 때문에 오류가 발생합니다 .
```

로컬 변수에 사용하는 변수 이름이 이미 전역 변수로 선언되어 있는 경우 해당 로컬 변수가 범위 안에 있는 동안에는 로컬 정의에 의해 전역 정의가 가려집니다. 이러한 경우에도 전역 변수는 함수 외부에서 계속 존재합니다. 예를 들어, 다음 코드에서는 `str1` 전역 문자열 변수를 만든 다음 `scopeTest()` 함수 내에 그와 동일한 이름으로 로컬 변수를 만듭니다. 함수 내의 `trace` 문은 로컬 변수의 값을 출력하지만 함수 외부의 `trace` 문은 전역 변수의 값을 출력합니다.

```
var str1:String = "Global";
function scopeTest ()
{
  var str1:String = "Local";
  trace(str1); // 로컬
}
scopeTest();
trace(str1); // 전역
```

C++ 및 Java의 변수와는 달리 ActionScript 변수에는 블록 레벨 범위가 없습니다. 코드 블록은 여는 중괄호({)와 닫는 중괄호(}) 사이의 모든 명령문 그룹입니다. C++ 및 Java 등 일부 프로그래밍 언어에서는 코드 블록 내에 선언된 변수를 해당 코드 블록 밖에서 사용할 수 없습니다. 이 범위 제한을 블록 레벨 범위라고 하며 ActionScript에는 이러한 제한이 없습니다. 코드 블록 내에 변수를 선언하면 해당 코드 블록뿐만 아니라 코드 블록이 속한 함수의 다른 모든 부분에서 해당 변수를 사용할 수 있습니다. 예를 들어, 다음 함수에는 여러 블록 범위에 정의된 변수가 포함되어 있습니다. 이러한 변수는 모두 함수 전체에서 사용할 수 있습니다.

```
function blockTest (testArray:Array)
{
  var numElements:int = testArray.length;
  if (numElements > 0)
  {
    var elemStr:String = "Element #";
    for (var i:int = 0; i < numElements; i++)
    {
      var valueStr:String = i + ": " + testArray[i];
      trace(elemStr + valueStr);
    }
    trace(elemStr, valueStr, i);   // 모두 정의되어 있음
  }
  trace(elemStr, valueStr, i); // numElements가 0보다 큰 경우 모두 정의됨
}

blockTest(["Earth", "Moon", "Sun"]);
```

블록 레벨 범위가 없다는 것은 함수 정의가 끝나기 전에 변수를 선언하기만 하면 해당 변수가 선언되기 이전에도 변수를 읽거나 변수에 쓸 수 있다는 것을 의미합니다. 이는 컴파일러에서 모든 변수 선언을 함수 맨 위로 이동하는 *호이스팅*이라는 기술에 의해 가능합니다. 예를 들어 다음 코드에서는 num 변수가 선언되기 전에 첫 번째 trace() 함수를 num 변수에 대해 호출하지만 정상적으로 컴파일됩니다.

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

그러나 컴파일러에서 대입문을 호이스팅하지는 않습니다. 따라서 num에 대한 첫 번째 trace() 호출의 결과가 Number 데이터 유형의 변수에 대한 기본값인 NaN(숫자 아님)이 되는 것입니다. 이는 다음 예제와 같이 변수가 선언되기 전에도 변수에 값을 지정할 수 있다는 것을 의미합니다.

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

기본값

*기본값*은 변수에 값을 설정하기 전에 해당 변수가 이미 보유하고 있는 값입니다. 변수에 처음으로 값을 설정하면 변수가 *초기화*됩니다. 그러나 변수를 선언하고 값을 설정하지 않으면 변수가 *초기화되지 않습니다*. 초기화되지 않은 변수의 값은 데이터 유형에 따라 달라집니다. 다음 표에는 변수의 기본값이 데이터 유형별로 설명되어 있습니다.

데이터 유형	기본값
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	`null`
uint	0
선언되지 않음(유형 약어 *에 해당)	`undefined`
사용자 정의 클래스를 포함한 기타 모든 클래스	`null`

Number 유형의 변수인 경우 기본 값은 `NaN`(숫자 아님)이며, 이는 숫자를 나타내지 않는 값을 의미하는 특수 값으로 IEEE-754 표준에 정의되어 있습니다.

변수를 선언하고 해당 데이터 유형을 선언하지 않은 경우 기본 데이터 유형인 *가 적용되며 이는 변수의 유형이 지정되지 않았다는 것을 의미합니다. 유형이 지정되지 않았고 값을 사용하여 초기화되지 않은 변수의 기본값은 `undefined`입니다.

Boolean, Number, int 및 uint 이외의 데이터 유형인 경우 초기화되지 않은 모든 변수의 기본값은 `null`입니다. 이는 Flash Player API에 의해 정의된 모든 클래스 및 사용자가 만든 모든 사용자 정의 클래스에 적용됩니다.

`null` 값은 Boolean, Number, int 또는 uint 유형의 변수에 사용할 수 없습니다. 이러한 변수에 `null` 값을 지정하려고 하면 이 값이 해당 데이터 유형에 대한 기본값으로 변환됩니다. Object 유형의 변수에는 `null`값을 지정할 수 있습니다. Object 유형의 변수에 `undefined` 값을 지정하려고 하면 이 값이 `null`로 변환됩니다.

Number 유형의 변수인지 확인할 때는 `isNaN()`이라는 특수한 최상위 함수를 사용합니다. 이 함수는 변수가 숫자가 아니면 부울 값으로 `true`를 반환하고 그렇지 않으면 `false`를 반환합니다.

데이터 유형

*데이터 유형*은 값의 집합을 정의합니다. 예를 들어, Boolean 데이터 유형은 `true`와 `false` 두 가지 값의 집합입니다. ActionScript 3.0에서는 Boolean 데이터 유형 외에도 String, Number 및 Array 등 보다 일반적으로 사용되는 여러 데이터 유형을 정의합니다. 클래스 또는 인터페이스를 사용하여 직접 데이터 유형을 정의하고 사용자 정의 값 집합을 정의할 수 있습니다. 프리미티브 값 또는 복합 값인지에 관계없이 ActionScript 3.0에서 모든 값은 객체입니다.

*프리미티브 값*은 Boolean, int, Number, String 및 uint 데이터 유형 중 하나에 속한 값입니다. 일반적으로 프리미티브 값을 사용하여 작업하면 복합 값을 사용하여 작업하는 것보다 속도가 빠릅니다. 이는 ActionScript에서 메모리와 속도를 최적화할 수 있는 특수 방식으로 프리미티브 값을 저장하기 때문입니다.

참고 **기술적인 세부 사항에 관심이 있는 독자를 위해 설명하자면 ActionScript 내부적으로는 프리미티브 값을 변경되지 않는 객체로 저장합니다. 프리미티브 값을 변경되지 않는 객체로 저장한다는 것은 참조에 의한 전달이 실제로는 값에 의한 전달과 동일함을 의미합니다. 일반적으로 값 자체의 크기보다 참조의 크기가 매우 작으므로 참조에 의한 전달을 통해 메모리 사용량이 줄어들고 실행 속도가 빨라집니다.**

*복합 값*은 프리미티브 값이 아닌 값입니다. 복합 값 집합을 정의하는 데이터 유형에는 Array, Date, Error, Function, RegExp, XML 및 XMLList 등이 있습니다.

많은 프로그래밍 언어에서 프리미티브 값과 래퍼 객체를 구분합니다. 예를 들어, Java에는 int 프리미티브 값과 이 값을 래핑하는 java.lang.Integer 클래스가 있습니다. Java 프리미티브 값은 객체가 아니지만 Java 프리미티브 값의 래퍼는 객체입니다. 이러한 차이점으로 인해 일부 연산에는 프리미티브 값이 유용하고 나머지 연산에는 래퍼 객체가 보다 적합합니다. ActionScript 3.0에서 프리미티브 값과 해당 래퍼 객체는 실제로 구분할 수 없습니다. 프리미티브 값을 포함한 모든 값이 객체입니다. Flash Player에서는 이러한 프리미티브 유형을 객체처럼 동작하지만 객체를 만드는 작업과 관련된 일반적인 오버헤드를 발생시키지 않는 특별한 경우로 간주합니다. 이는 다음 두 코드 행이 동일하다는 것을 의미합니다.

```
var someInt:int = 3;
var someInt:int = new int(3);
```

위에 나열된 프리미티브 및 복합 데이터 유형은 모두 ActionScript 3.0 기본 클래스에 의해 정의됩니다. 기본 클래스를 사용하면 `new` 연산자 대신 리터럴 값을 사용하여 객체를 만들 수 있습니다. 예를 들어 다음과 같이 리터럴 값 또는 Array 클래스 생성자를 사용하여 배열을 만들 수 있습니다.

```
var someArray:Array = [1, 2, 3]; // 리터럴 값
var someArray:Array = new Array(1,2,3); // Array 생성자
```

유형 검사

유형 검사는 컴파일 타임이나 런타임에 수행할 수 있습니다. C++ 및 Java와 같이 정적으로 유형이 지정되는 언어는 컴파일 타임에 유형 검사를 수행합니다. Smalltalk 및 Python과 같이 동적으로 유형이 지정되는 언어는 런타임에 유형 검사를 처리합니다. 동적으로 유형이 지정되는 언어인 ActionScript 3.0의 경우 런타임에 유형 검사를 수행하지만 *Strict 모드*라는 특수 컴파일러 모드에서의 컴파일 타임 유형 검사도 지원합니다. Strict 모드에서는 컴파일 타임과 런타임에 모두 유형 검사가 수행되지만, Standard 모드에서는 런타임에만 유형 검사가 수행됩니다.

동적으로 유형이 지정되는 언어를 사용하면 코드를 구성할 때 상당한 융통성을 누릴 수 있지만 런타임에 유형 오류가 출력되는 것을 감수해야 합니다. 정적으로 유형이 지정되는 언어를 사용하면 컴파일 타임에 유형 오류를 확인할 수 있지만 컴파일러에게 유형 정보를 제공해야 하는 부담을 지게 됩니다.

컴파일 타임 유형 검사

프로젝트 규모가 커지면 일반적으로 데이터 유형 유연성보다 유형 오류를 가능한 빨리 포착하는 것이 중요해지므로 대형 프로젝트에서는 컴파일 타임 유형 검사가 자주 사용됩니다. 이러한 이유 때문에 Adobe Flash CS3 Professional 및 Adobe Flex Builder 2의 ActionScript 컴파일러가 Strict 모드에서 실행되도록 기본 설정되어 있습니다.

컴파일 타임 유형 검사를 제공하려면 컴파일러에서 코드의 변수 또는 표현식에 대한 데이터 유형 정보를 알고 있어야 합니다. 변수의 데이터 유형을 명시적으로 선언하려면 변수 이름에 콜론 연산자(:)를 추가하고 그 뒤에 데이터 유형을 접미어로 추가합니다. 데이터 유형을 매개 변수와 연결하려면 콜론 연산자와 데이터 유형을 차례로 사용합니다. 예를 들어 다음 코드에서는 `xParam` 매개 변수에 데이터 유형 정보를 추가하고 명시적 데이터 유형을 사용하여 `myParam` 변수를 선언합니다.

```
function runtimeTest(xParam:String)
{
  trace(xParam);
}
var myParam:String = “hello”;
runtimeTest(myParam);
```

Strict 모드의 ActionScript 컴파일러에서는 컴파일러 오류에 따라 유형 불일치를 보고합니다. 예를 들어, 다음 코드에서는 Object 유형의 xParam 함수 매개 변수를 선언하지만 이후 해당 매개 변수에 String 및 Number 유형 값을 지정하려고 합니다. 이로 인해 Strict 모드에서 컴파일러 오류가 발생합니다.

```
function dynamicTest(xParam:Object)
{
  if (xParam is String)
  {
    var myStr:String = xParam; // Strict 모드에서 컴파일러 오류가 발생합니다.
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam; // Strict 모드에서 컴파일러 오류가 발생합니다.
    trace("Number: " + myNum);
  }
}
```

그러나 Strict 모드에서도 대입문의 우변을 유형이 지정되지 않은 상태로 두면 컴파일 타임 유형 검사에서 해당 대입문을 제외할 수 있습니다. 유형 약어를 생략하거나 특수 별표(*) 유형 약어를 사용하여 유형이 지정되지 않은 변수 또는 표현식을 표시할 수 있습니다. 예를 들어, 이전 예제를 수정하여 xParam 매개 변수의 유형 약어를 생략하면 Strict 모드에서 코드가 컴파일됩니다.

```
function dynamicTest(xParam)
{
  if (xParam is String)
  {
    var myStr:String = xParam;
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam;
    trace("Number: " + myNum);
  }
}
dynamicTest(100)
dynamicTest("one hundred");
```

런타임 유형 검사

ActionScript 3.0에서는 Strict 모드 또는 Standard 모드 중 어느 모드에서 컴파일했는지와 관계없이 런타임 유형 검사가 수행됩니다. 배열 인수가 필요한 함수에 값 3이 전달되는 경우를 생각해 봅시다. 값 3이 Array 데이터 유형과 호환되지 않으므로 Strict 모드의 컴파일러에서 오류가 발생합니다. Strict 모드를 사용하지 않고 Standard 모드에서 실행하는 경우 컴파일러에서는 유형 불일치에 대한 오류가 발생하지 않지만 Flash Player에서 런타임 유형 검사를 수행하면 런타임 오류가 발생합니다.

다음 예제에서는 필요한 Array 인수 대신 값 3이 전달된 `typeTest()` 함수를 보여 줍니다. 값 3이 매개 변수의 선언된 데이터 유형(Array)의 멤버가 아니기 때문에 Standard 모드에서 런타임 오류가 발생합니다.

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// Standard 모드에서 런타임 오류가 발생합니다.
```

Strict 모드에서 작업하는 경우에도 런타임 유형 오류가 발생할 수 있습니다. Strict 모드에서 유형이 지정되지 않은 변수를 사용하면 컴파일 타임 유형 검사에서 해당 변수가 제외되므로 런타임에 유형 오류가 발생할 수도 있습니다. 유형이 지정되지 않은 변수를 사용하는 경우 유형 검사는 취소되지 않고 런타임까지 연기됩니다. 예를 들어 이전 예제의 `myNum` 변수에 선언된 데이터 유형이 없는 경우 컴파일러에서는 유형 불일치를 감지할 수 없지만 Flash Player에서는 런타임 오류가 발생합니다. 이는 Flash Player에서 대입문의 결과 값인 3으로 설정된 `myNum`의 런타임 값을 Array 데이터 유형으로 설정된 `xParam`의 유형과 비교하기 때문입니다.

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// ActionScript 3.0에서 런타임 오류가 발생합니다.
```

또한 런타임 유형 검사의 경우 컴파일 타임 검사에 비해 상속의 사용을 덜 엄격하게 검사합니다. Standard 모드에서 유형 검사를 런타임으로 연기하면 하위 클래스를 *업캐스팅*한 경우에도 하위 클래스의 속성을 참조할 수 있습니다. 기본 클래스를 사용하여 클래스 인스턴스의 유형을 선언하고, 하위 클래스를 사용하여 클래스 인스턴스를 인스턴스화하면 업캐스팅이 일어납니다. 예를 들어 확장할 수 있는 ClassBase 클래스를 만들 수 있습니다. `final` 특성이 있는 클래스는 확장할 수 없습니다.

```
class ClassBase
{
}
```

이어서 다음과 같이 someString 속성 하나가 있는 ClassExtender라는 ClassBase의 하위 클래스를 만들 수 있습니다.

```
class ClassExtender extends ClassBase
{
  var someString:String;
}
```

두 클래스를 사용하면 ClassBase 데이터 유형으로 선언되고 ClassExtender 생성자로 인스턴스화되는 클래스 인스턴스를 만들 수 있습니다. 기본 클래스에는 하위 클래스에 없는 속성 또는 메서드가 포함되어 있지 않기 때문에 업캐스팅은 안전한 작업으로 간주됩니다.

```
var myClass:ClassBase = new ClassExtender();
```

그러나 하위 클래스에는 기본 클래스에 없는 속성 또는 메서드가 포함되어 있습니다. 예를 들어, ClassExtender 클래스에는 ClassBase 클래스에 없는 someString 속성이 포함되어 있습니다. ActionScript 3.0 Standard 모드에서 다음 예제와 같이 컴파일 타임 오류를 생성하지 않고 myClass 인스턴스를 사용하여 이 속성을 참조할 수 있습니다.

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// ActionScript 3.0 Standard 모드에서 오류가 발생하지 않습니다 .
```

is 연산자

ActionScript 3.0에 새로 추가된 is 연산자를 사용하면 변수 또는 표현식이 지정된 데이터 유형의 멤버인지 여부를 테스트할 수 있습니다. 이전 버전의 ActionScript에서 instanceof 연산자가 이 기능을 제공했지만 ActionScript 3.0에서는 instanceof 연산자를 데이터 유형 멤버를 테스트하는 데 사용할 수 없습니다. x instanceof y 표현식으로는 단지 x의 프로토타입 체인에 y가 존재하는지만 확인할 수 있으며, ActionScript 3.0의 프로토타입 체인을 통해서는 완전한 상속 계층 구조에 액세스할 수 없으므로 유형 검사를 수동으로 수행하려면 instanceof 연산자 대신 is 연산자를 사용해야 합니다.

is 연산자는 정확한 상속 계층 구조를 검사하며 객체가 특정 클래스의 인스턴스인지 뿐만 아니라 객체가 특정 인터페이스를 구현하는 클래스의 인스턴스인지 여부를 확인하는 데 사용됩니다. 다음 예제에서는 mySprite라는 Sprite 클래스의 인스턴스를 만들고 is 연산자를 사용하여 mySprite가 Sprite와 DisplayObject 클래스의 인스턴스인지 및 IEventDispatcher 인터페이스를 구현하는지 여부를 테스트합니다.

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite);                  // true
trace(mySprite is DisplayObject);           // true
trace(mySprite is IEventDispatcher);        // true
```

is 연산자는 상속 계층 구조를 검사하고 mySprite가 Sprite 및 DisplayObject 클래스와 호환되는지 보고합니다. Sprite 클래스는 DisplayObject 클래스의 하위 클래스입니다. 또한 is 연산자는 mySprite가 IEventDispatcher를 구현하는 클래스에서 상속되는지 여부를 확인합니다. Sprite 클래스가 IEventDispatcher 인터페이스를 구현하는 EventDispatcher 클래스에서 상속되므로 is 연산자는 mySprite가 동일한 인터페이스를 구현하고 있음을 정확하게 보고합니다.

다음 예제에서는 이전 예제와 동일한 테스트를 보여 주지만 is 연산자 대신 instanceof를 사용합니다. instanceof 연산자는 mySprite가 Sprite 또는 DisplayObject의 인스턴스인지를 정확하게 식별하지만 mySprite에서 IEventDispatcher 인터페이스를 구현하는지 여부를 테스트하기 위해 사용하는 경우에는 false를 반환합니다.

```
trace(mySprite instanceof Sprite);           // true
trace(mySprite instanceof DisplayObject);    // true
trace(mySprite instanceof IEventDispatcher); // false
```

as 연산자

ActionScript 3.0에 새로 추가된 as 연산자를 사용하면 표현식이 지정된 데이터 유형의 멤버인지 여부를 확인할 수 있습니다. is 연산자와 달리 as 연산자는 부울 값을 반환하지 않습니다. as 연산자는 true 대신 표현식의 값을 반환하고 false 대신 null을 반환합니다. 다음 예제에서는 Sprite 인스턴스가 DisplayObject, IEventDispatcher 및 Number 데이터 유형의 멤버인지 여부를 확인할 때 is 연산자 대신 as 연산자를 사용하는 경우에 대한 결과를 보여 줍니다.

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [object Sprite]
trace(mySprite as DisplayObject); // [object Sprite]
trace(mySprite as IEventDispatcher); // [object Sprite]
trace(mySprite as Number);   // null
```

as 연산자를 사용하는 경우 오른쪽 피연산자는 데이터 유형이어야 합니다. 오른쪽 피연산자로 데이터 유형이 아닌 표현식을 사용하려고 하면 오류가 발생합니다.

동적 클래스

동적 클래스는 속성 및 메서드를 추가하거나 변경하여 런타임에 변경할 수 있는 객체를 정의합니다. String 클래스와 같이 동적이지 않은 클래스는 *봉인된* 클래스입니다. 런타임에 속성 또는 메서드를 봉인된 클래스에 추가할 수 없습니다.

클래스를 선언할 때 dynamic 특성을 사용하여 동적 클래스를 만듭니다. 예를 들어, 다음 코드에서는 Protean이라는 동적 클래스를 만듭니다.

```
dynamic class Protean
{
  private var privateGreeting:String = "hi";
  public var publicGreeting:String = "hello";
  function Protean()
```

```
  {
    trace("Protean instance created");
  }
}
```

이어서 Protean 클래스의 인스턴스를 인스턴스화하는 경우 클래스 정의 외부에 있는 인스턴스에 속성 또는 메서드를 추가할 수 있습니다. 예를 들어, 다음 코드에서는 Protean 클래스의 인스턴스를 만들고 aString 속성과 aNumber 속성을 해당 인스턴스에 추가합니다.

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // 테스트 3
```

동적 클래스의 인스턴스에 추가한 속성은 런타임 엔터티이므로 모든 유형 검사는 런타임에 수행됩니다. 이러한 방법으로 추가한 속성에 유형 약어를 추가할 수 없습니다.

함수를 정의하고 해당 함수를 myProtean 인스턴스의 속성에 추가하여 myProtean 인스턴스에 메서드를 추가할 수도 있습니다. 다음 코드에서는 trace 문을 traceProtean() 메서드로 이동합니다.

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
  trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // 테스트 3
```

그러나 이렇게 만든 메서드에서는 Protean 클래스의 모든 전용 속성 또는 메서드에 액세스하지 못합니다. 또한 Protean 클래스의 공용 속성 또는 메서드에 대한 참조를 this 키워드 또는 클래스 이름으로 정규화해야 합니다. 다음 예제에서는 Protean 클래스의 전용 및 공용 변수에 액세스하려고 하는 traceProtean() 메서드를 보여 줍니다.

```
myProtean.traceProtean = function ()
{
  trace(myProtean.privateGreeting); // undefined
  trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

데이터 유형 설명

프리미티브 데이터 유형에는 Boolean, int, Null, Number, String, uint 및 void 등이 포함됩니다. 또한 ActionScript 기본 클래스에서 Object, Array, Date, Error, Function, RegExp, XML 및 XMLList 등의 복합 데이터 유형을 정의합니다.

Boolean 데이터 유형

Boolean 데이터 유형은 `true`와 `false` 두 가지 값으로 구성됩니다. 그 외 다른 값은 Boolean 유형의 변수에 사용할 수 없습니다. 선언되었지만 초기화되지 않은 부울 변수의 기본값은 `false`입니다.

int 데이터 유형

int 데이터 유형은 내부적으로 32비트의 정수로 저장되며 -2,147,483,648(-2^{31})부터 2,147,483,647($2^{31} - 1$)까지의 정수 집합(-2,147,483,648 및 2,147,483,647 포함)으로 구성됩니다. 이전 버전의 ActionScript에서는 정수와 부동 소수점 숫자에 모두 사용되는 Number 데이터 유형만 제공했습니다. ActionScript 3.0에서는 이제 32비트의 부호가 있거나 없는 정수에 대한 저수준의 시스템 유형에 액세스할 수 있습니다. 변수에서 부동 소수점 숫자를 사용하지 않는 경우 Number 데이터 유형 대신 int 데이터 유형을 사용하는 것이 보다 빠르고 효율적입니다.

최소 및 최대 int 값의 범위를 벗어나는 정수 값인 경우 Number 데이터 유형을 사용하여 양수와 음수 각각 9,007,199,254,740,992(53비트 정수 값) 사이의 값을 처리할 수 있습니다.
int 데이터 유형의 변수에 대한 기본값은 0입니다.

Null 데이터 유형

Null 데이터 유형에는 `null` 값만 포함됩니다. 이는 Object 클래스를 포함하여 복합 데이터 유형을 정의하는 모든 클래스와 String 데이터 유형의 기본값입니다. Boolean, Number, int 및 uint 같은 기타 프리미티브 데이터 유형에는 `null` 값이 포함되어 있지 않습니다. Boolean, Number, int 또는 uint 유형의 변수에 `null`을 지정하려고 하면 Flash Player에서 `null` 값을 적절한 기본값으로 변환합니다. 이 데이터 유형은 유형 약어로 사용할 수 없습니다.

Number 데이터 유형

ActionScript 3.0에서 Number 데이터 유형은 정수, 부호 없는 정수, 부동 소수점 숫자 등을 나타낼 수 있습니다. 그러나 성능을 최대화하려면 32비트 `int` 및 `uint` 유형이 저장할 수 있는 값보다 큰 정수 값 또는 부동 소수점 숫자에만 Number 데이터 유형을 사용해야 합니다. 부동 소수점 숫자를 저장하려면 숫자 안에 소수점을 넣습니다. 소수점을 생략하면 숫자가 정수로 저장됩니다.

Number 데이터 유형에서는 이진 부동 소수점 산술에 대한 IEEE 표준(IEEE-754)에 지정된 64비트 배정밀도 형식을 사용합니다. 이 표준에는 64비트를 사용하여 부동 소수점 숫자를 저장하는 방법이 규정되어 있습니다. 1비트는 숫자가 양수 또는 음수인지 지정하는 데 사용됩니다. 11비트는 2를 밑수로 하는 지수를 저장하는 데 사용됩니다. 남은 52비트는 지수에 따라 거듭제곱되는 숫자인 *유효 숫자*(또는 *가수*)를 저장하는 데 사용됩니다.

Number 데이터 유형에서는 지수를 저장하는 데 일부 비트를 사용하므로 모든 비트가 유효 숫자를 저장하는 데 사용되는 경우에 비해 매우 큰 부동 소수점 숫자를 저장할 수 있습니다. 예를 들어, Number 데이터 유형에서 유효 숫자를 저장하는 데 64비트를 모두 사용하는 경우, 저장할 수 있는 가장 큰 숫자는 2^{65} - 1입니다. 11비트를 사용하여 지수를 저장하면 Number 데이터 유형에서 유효 숫자를 2^{1023}으로 거듭제곱할 수 있습니다.

Number 유형에서 나타낼 수 있는 최소 및 최대 값은 `Number.MAX_VALUE` 및 `Number.MIN_VALUE`라는 Number 클래스의 정적 속성에 저장됩니다.

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

Number 데이터 유형의 경우 숫자의 범위는 방대하지만 정밀도는 떨어집니다. Number 데이터 유형에서는 유효 숫자를 저장하는 데 52비트를 사용하므로 정확하게 나타내기 위해 52비트 이상이 필요한 숫자(예: 분수 1/3)는 근삿값만 표현할 수 있습니다. 응용 프로그램에서 절대 정밀도의 십진수가 필요한 경우 이진 부동 소수점 산술과 대립되는 십진 부동 소수점 산술을 구현하는 소프트웨어를 사용해야 합니다.

Number 데이터 유형을 사용하여 정수 값을 저장하면 유효 숫자의 52비트만 사용됩니다. Number 데이터 유형에서는 해당 52비트 및 특수 은폐 비트를 사용하여 -9,007,199,254,740,992(-2^{53})부터 9,007,199,254,740,992(2^{53})까지의 정수를 나타냅니다.

Flash Player에서는 `NaN` 값을 `Number` 유형의 변수에 대한 기본값으로 사용할 뿐만 아니라 숫자를 반환해야 하지만 이를 반환하지 않는 모든 연산의 결과로도 사용합니다. 예를 들어, 음수의 제곱근을 계산하려고 하면 결과는 `NaN`이 됩니다. 기타 특수 Number 값에 *양의 무한대*와 *음의 무한대*가 포함됩니다.

예

제수가 `0`인 경우 `0`으로 나눈 결과도 `NaN`입니다. 피제수가 양수인 경우 `0`으로 나누면 `무한대`가 되고, 피제수가 음수인 경우 `0`으로 나누면 `음의 무한대`가 됩니다.

String 데이터 유형

String 데이터 유형은 16비트 문자열을 나타냅니다. 문자열은 내부적으로 UTF-16 포맷을 사용하여 유니코드 문자로 저장합니다. 문자열은 Java 프로그래밍 언어와 마찬가지로 변경할 수 없는 값입니다. String 값에 대한 연산은 해당 문자열의 새 인스턴스를 반환합니다. String 데이터 유형으로 선언되는 변수의 기본값은 `null`입니다. `null` 값과 빈 문자열(`""`) 모두 문자가 없다는 것을 나타내지만 서로 다릅니다.

uint 데이터 유형

uint 데이터 유형은 내부적으로 32비트의 부호 없는 정수로 저장되며 0부터 4,294,967,295(2^{32} - 1)까지의 정수 집합(0 및 4,294,967,295 포함)으로 구성됩니다. 음이 아닌 정수를 호출하는 특수한 경우에는 uint 데이터 유형을 사용합니다. 예를 들어 int 데이터 유형에는 색상 값 처리에 적합하지 않은 내부 부호 비트가 포함되어 있기 때문에 픽셀의 색상 값을 나타내려면 uint 데이터 유형을 사용해야 합니다. 최대 uint 값보다 큰 정수 값의 경우에는 53비트 정수 값을 처리할 수 있는 Number 데이터 유형을 사용합니다. uint 데이터 유형의 변수에 대한 기본값은 0입니다.

void 데이터 유형

void 데이터 유형에는 `undefined` 값만 포함됩니다. 이전 버전의 ActionScript에서 Object 클래스의 인스턴스에 대한 기본값으로 `undefined`를 사용했습니다. ActionScript 3.0에서 Object 인스턴스에 대한 기본값은 `null`입니다. Object 클래스의 인스턴스에 `undefined` 값을 지정하려고 하면 Flash Player에서 이 값을 `null`로 변환합니다. `undefined` 값은 유형이 지정되지 않은 변수에만 지정할 수 있습니다. 유형이 지정되지 않은 변수는 유형 약어가 없거나 유형 약어에 별표(*) 기호를 사용하는 변수입니다. `void`는 반환 유형 약어로만 사용할 수 있습니다.

Object 데이터 유형

Object 데이터 유형은 Object 클래스에 의해 정의됩니다. Object 클래스는 ActionScript에서 모든 클래스 정의에 대한 기본 클래스 역할을 합니다. ActionScript 3.0 버전의 Object 데이터 유형은 세 가지 면에서 이전 버전의 Object 데이터 유형과 다릅니다. 첫 번째, Object 데이터 유형은 더 이상 유형 약어가 없는 변수에 지정되는 기본 데이터 유형이 아닙니다. 두 번째, Object 데이터 유형에는 Object 인스턴스의 기본값이었던 `undefined` 값이 더 이상 포함되지 않습니다. 세 번째, ActionScript 3.0에서는 Object 클래스의 인스턴스에 대한 기본값이 `null`입니다.

이전 버전의 ActionScript에서 유형 약어가 없는 변수는 자동으로 Object 데이터 유형으로 지정되었습니다. ActionScript 3.0에는 유형이 지정되지 않은 변수라는 개념이 도입되면서 이전 버전과는 다르게 처리됩니다. 이제 유형 약어가 없는 변수는 유형이 지정되지 않은 변수로 간주됩니다. 변수의 유형을 지정하지 않으려는 자신의 의도를 코드를 읽는 사람에게 분명히 밝히려면 유형 약어로 새 별표(*) 기호를 사용할 수 있습니다. 이는 유형 약어를 생략하는 것과 같습니다. 다음 예제에서는 유형이 지정되지 않은 변수 `x`를 선언하는 동일한 두 명령문을 보여 줍니다.

```
var x
var x:*
```

유형이 지정되지 않은 변수에만 undefined 값을 저장할 수 있습니다. 데이터 유형이 있는 변수에 undefined 값을 지정하려고 하면 Flash Player에서 undefined 값을 해당 데이터 유형의 기본값으로 변환합니다. Object 데이터 유형의 인스턴스인 경우 기본값은 null이며, 이는 Object 인스턴스에 undefined를 지정하려고 하면 Flash Player에서 undefined 값을 null로 변환한다는 것을 의미합니다.

유형 변환

어떤 값이 다른 데이터 유형의 값으로 변형되면 유형 변환이 발생한다고 합니다. 유형 변환은 *암시적* 또는 *명시적*으로 수행될 수 있습니다. *강제 형 변환*이라고도 하는 암시적 변환이 런타임에 Flash Player에서 수행되는 경우도 있습니다. 예를 들어, Boolean 데이터 유형에 값 2를 지정하면 변수에 값을 지정하기 전에 Flash Player에서 값 2를 부울 값인 true로 변환합니다. 컴파일러가 특정 데이터 유형의 변수를 다른 데이터 유형의 변수로 처리하도록 코드에서 지정하면 *형 변환*이라는 명시적 변환이 발생합니다. 프리미티브 값이 포함되어 있으면 형 변환에 의해 실제로 한 데이터 유형에서 다른 데이터 유형으로 값이 변환됩니다. 객체를 다른 유형으로 변환하려면 객체 이름을 괄호로 묶고 그 앞에 새 유형의 이름을 붙여야 합니다. 예를 들어 다음 코드에서는 부울 값을 전달 받아 정수로 변환합니다.

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

암시적 변환

런타임에 암시적 변환이 발생하는 몇 가지 경우는 다음과 같습니다.

- 대입문인 경우
- 값이 함수 인수로 전달되는 경우
- 함수에서 값이 반환되는 경우
- 추가(+) 연산자와 같은 특정 연산자를 사용하는 식인 경우

사용자 정의 유형의 경우 변환할 값이 대상 클래스 또는 대상 클래스에서 파생된 클래스의 인스턴스이면 암시적 변환이 이루어집니다. 암시적 변환에 실패하면 오류가 발생합니다. 예를 들어, 다음 코드에는 성공적인 암시적 변환과 실패한 암시적 변환이 포함되어 있습니다.

```
class A {}
class B extends A {}

var objA:A = new A();
var objB:B = new B();
var arr:Array = new Array();

objA = objB; // 변환이 성공합니다 .
objB = arr; // 변환이 실패합니다 .
```

프리미티브 유형의 경우 명시적 변환 함수에서 호출하는 내부 변환 알고리즘과 동일한 알고리즘을 호출하여 암시적 변환을 처리합니다. 다음 단원에서 이러한 프리미티브 유형 변환에 대해 자세히 설명합니다.

명시적 변환

Strict 모드에서 컴파일할 때 유형 불일치로 인해 컴파일 타임 오류가 발생하지 않도록 하려면 명시적 변환 또는 형 변환을 사용하는 것이 도움이 됩니다. 강제 형 변환이 런타임에 값을 올바르게 변환한다는 것을 알고 있는 경우를 가정해 봅시다. 예를 들어, 양식에서 전달된 데이터를 처리하는 경우 특정 문자열 값을 숫자 값으로 변환하기 위해 강제 형 변환을 사용할 수 있습니다. 다음 코드는 Standard 모드에서 올바르게 실행되지만 컴파일 타임에 오류가 발생합니다.

```
var quantityField:String = "3";
var quantity:int = quantityField; // Strict 모드에서 컴파일 시 오류가 발생합니다.
```

계속해서 Strict 모드를 사용하면서 문자열이 정수로 변환되도록 하려면 다음과 같이 명시적 변환을 사용할 수 있습니다.

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // 명시적 변환이 성공합니다.
```

int, uint 및 Number로 형 변환

모든 데이터 유형을 int, uint 및 Number 등의 숫자 유형 중 하나로 변환할 수 있습니다. 어떤 이유로 Flash Player에서 숫자를 변환할 수 없는 경우 int 및 uint 데이터 유형에 기본값으로 0이 지정되고 Number 데이터 유형에는 기본값으로 `NaN`이 지정됩니다. 부울 값을 숫자로 변환하면 `true`는 값 1이 되며 `false`는 값 0이 됩니다.

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

숫자만 포함된 문자열 값을 숫자 유형 중 하나로 변환할 수 있습니다. 음수 형태의 문자열 또는 16진수 값(예: `0x1A`)을 나타내는 문자열도 숫자 유형으로 변환할 수 있습니다. 변환 과정에서 문자열 값에 있는 선행 또는 후행 공백 문자는 무시됩니다. `Number()`를 사용하여 부동 소수점 숫자 형태의 문자열을 변환할 수도 있습니다. 소수점이 포함된 문자열을 `uint()` 및 `int()`에 전달하면 소수점과 그 뒤의 문자를 잘라내고 정수를 반환합니다. 예를 들어 다음 문자열 값을 숫자로 변환할 수 있습니다.

```
trace(uint("5"));      // 5
trace(uint("-5"));     // 4294967291. MAX_VALUE에서 처음부터 다시 시작됩니다.
trace(uint(" 27 "));   // 27
trace(uint("3.7"));    // 3
trace(int("3.7"));     // 3
trace(int("0x1A"));    // 26
trace(Number("3.7")); // 3.7
```

숫자가 아닌 문자가 포함된 문자열 값을 `int()` 또는 `uint()`로 변환하면 0이 반환되고 `Number()`로 변환하면 `NaN`이 반환됩니다 변환 과정에서 선행 또는 후행 공백 문자는 무시되지만 문자열에 두 개의 숫자를 구분하는 공백이 있는 경우 0 또는 `NaN`을 반환합니다.

```
trace(uint("5a"));     // 0
trace(uint("ten"));    // 0
trace(uint("17 63")); // 0
```

ActionScript 3.0에서 `Number()` 함수는 더 이상 8진수 또는 밑수가 8인 숫자를 지원하지 않습니다. ActionScript 2.0에서 `Number()` 함수에 0으로 시작되는 문자열을 제공하면 숫자가 8진수로 해석된 후 10진수로 변환됩니다. ActionScript 3.0에서 `Number()` 함수는 문자열 맨 앞에 있는 0을 무시하기 때문에 이전 버전에서와 같이 수행되지 않습니다. 예를 들어 다음 코드를 서로 다른 버전의 ActionScript를 사용하여 컴파일하면 각기 다른 결과가 출력됩니다.

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

특정 숫자 유형의 값이 다른 숫자 유형의 변수에 지정된 경우에는 변환할 필요가 없습니다. Strict 모드에서도 숫자 유형이 다른 숫자 유형으로 암시적으로 변환됩니다. 이는 경우에 따라 유형 범위를 벗어나면 예기치 못한 값이 발생할 수 있다는 것을 의미합니다. 경우에 따라 예기치 못한 값이 생성될 수 있지만 다음 예제는 Strict 모드에서 모두 컴파일됩니다.

```
var myUInt:uint = -3; // uint 변수에 int/Number 값을 지정합니다.
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Number 변수에 int/uint 값을 지정합니다.
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // uint 변수에 Number 값을 지정합니다.
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // int 변수에 uint/Number 값을 지정합니다.
trace(myInt); // -2147483648
```

다음 표에는 다른 데이터 유형에서 Number, int 또는 uint 데이터 유형으로의 변환 결과가 요약되어 있습니다.

데이터 유형 또는 값	Number, int 또는 uint로 변환한 결과
Boolean	값이 `true`이면 1로 변환되고 그렇지 않으면 0으로 변환됩니다.
Date	Date 객체의 내부 표현 즉, 표준시 1970년 1월 1일 자정 이후 경과된 밀리초로 변환됩니다.
`null`	0
Object	`null`인 인스턴스를 Number로 변환하면 `NaN`이 반환되고 그 외의 경우에는 0이 반환됩니다.
String	Flash Player에서 문자열을 숫자로 변환할 수 있는 경우 숫자가 반환되고, 그렇지 않은 경우 Number로 변환하면 `NaN`이 반환되고 int 또는 uint로 변환하면 0이 반환됩니다.
`undefined`	Number로 변환하면 `NaN`이 반환되고 int 또는 uint로 변환하면 0이 반환됩니다.

Boolean으로 변환

숫자 데이터 유형(uint, int 및 Number)에서 Boolean으로 변환하면, 숫자 값이 0인 경우 `false`가 반환되고 그렇지 않으면 `true`가 반환됩니다. Number 데이터 유형의 경우 `NaN` 값 역시 `false`가 됩니다. 다음 예제에서는 숫자 -1, 0 및 1을 변환한 결과를 보여 줍니다.

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
  trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

이 예제의 출력에서 숫자 0만 `false` 값을 반환하는 것을 확인할 수 있습니다.

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

문자열 값에서 Boolean으로 변환하는 경우 문자열이 `null`이거나 빈 문자열(`""`)이면 `false`를 반환합니다. 그렇지 않으면 `true`를 반환합니다.

```
var str1:String;        // 초기화되지 않은 문자열은 null 입니다 .
trace(Boolean(str1));  // false

var str2:String = "";  // 빈 문자열
trace(Boolean(str2));  // false

var str3:String = " "; // 공백만
trace(Boolean(str3));  // true
```

Object 클래스의 인스턴스에서 Boolean으로 변환하는 경우 인스턴스가 null이면 false를 반환하고 그렇지 않으면 true를 반환합니다.

```
var myObj:Object;       // 초기화되지 않은 객체는 null 입니다 .
trace(Boolean(myObj)); // false

myObj = new Object();  // 인스턴스화합니다 .
trace(Boolean(myObj)); // true
```

Strict 모드에서 모든 데이터 유형의 값을 Boolean으로 변환하지 않고 부울 변수에 지정할 수 있다는 점에서 부울 변수는 특수하게 처리되는 변수입니다. Strict 모드에서도 모든 데이터 유형에서 Boolean 데이터 유형으로 암시적으로 강제 형 변환합니다. 즉, 대부분의 다른 모든 데이터 유형과 달리 Strict 모드 오류를 방지하기 위해 Boolean으로 변환할 필요가 없습니다. 다음 예제는 Strict 모드에서 모두 컴파일되고 런타임에 예상대로 작동됩니다.

```
var myObj:Object = new Object();  // 인스턴스화합니다 .
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

다음 표에는 다른 데이터 유형에서 Boolean 데이터 유형으로 변환한 결과가 요약되어 있습니다.

데이터 유형 또는 값	Boolean로 변환한 결과
String	값이 null 또는 빈 문자열("")이면 false를 반환하고 그렇지 않으면 true를 반환합니다.
null	false
Number, int 또는 uint	값이 NaN 또는 0이면 false를 반환하고 그렇지 않으면 true를 반환합니다.
Object	인스턴스가 null이면 false를 반환하고 그렇지 않으면 true를 반환합니다.

String으로 변환

모든 숫자 데이터 유형에서 String 데이터 유형으로 변환하면 숫자의 문자열 표현을 반환합니다. 부울 값에서 String 데이터 유형으로 변환하면 값이 true인 경우 "true" 문자열을 반환하고 값이 false인 경우 "false" 문자열을 반환합니다.

Object 클래스의 인스턴스에서 String 데이터 유형으로 변환하면 인스턴스가 null인 경우 "null" 문자열을 반환합니다. 그렇지 않은 경우, Object 클래스에서 String 유형으로 변환하면 "[object Object]" 문자열을 반환합니다.

Array 클래스의 인스턴스에서 String으로 변환하면 모든 배열 요소가 쉼표로 구분된 목록으로 구성된 문자열을 반환합니다. 예를 들어, 다음과 같이 String 데이터 유형으로 변환하면 세 가지 배열 요소가 모두 포함되어 있는 하나의 문자열을 반환합니다.

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary,secondary,tertiary
```

Date 클래스의 인스턴스에서 String으로 변환하면 인스턴스에 포함된 날짜의 문자열 표현을 반환합니다. 예를 들어, 다음 예제에서는 Date 클래스 인스턴스의 문자열 표현(태평양 일광 절약 시간으로 결과 표시)을 반환합니다.

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Fri Jul 1 00:00:00 GMT-0700 2005
```

다음 표에는 다른 데이터 유형에서 String 데이터 유형으로 변환한 결과가 요약되어 있습니다.

데이터 유형 또는 값	문자열로 변환한 결과
Array	모든 배열 요소로 구성된 문자열을 반환합니다.
Boolean	`"true"` 또는 `"false"`를 반환합니다.
Date	Date 객체의 문자열 표현을 반환합니다.
`null`	`"null"`
Number, int 또는 uint	숫자의 문자열 표현을 반환합니다.
Object	인스턴스가 null이면 `"null"`을 반환하고 그렇지 않으면 `"[object Object]"`를 반환합니다.

구문

언어의 구문에는 실행 코드를 작성할 때 따라야 하는 규칙 집합이 정의되어 있습니다.

대/소문자 구분

ActionScript 3.0에서는 대/소문자를 구분합니다. 대/소문자만 다르고 이름이 같은 식별자는 서로 다른 식별자로 간주합니다. 예를 들어 다음 코드에서는 두 개의 서로 다른 변수를 만듭니다.

```
var num1:int;
var Num1:int;
```

도트 구문

도트 연산자(.)는 객체의 속성 및 메서드에 액세스하는 방법을 제공합니다. 도트 구문을 사용하면 도트 연산자 및 속성 또는 메서드 이름 앞에 인스턴스 이름을 사용하여 클래스 속성 또는 메서드를 참조할 수 있습니다. 예를 들어 다음과 같은 클래스 정의를 생각해 봅시다.

```
class DotExample
{
  public var prop1:String;
  public function method1():void {}
}
```

도트 구문을 사용하면 다음 코드에서 만든 인스턴스 이름을 사용하여 `prop1` 속성 및 `method1()` 메서드에 액세스할 수 있습니다.

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = “hello”;
myDotEx.method1();
```

패키지를 정의할 때 도트 구문을 사용할 수 있습니다. 도트 연산자를 사용하여 중첩된 패키지를 참조합니다. 예를 들어, 플래시 패키지 내에 중첩된 이벤트 패키지에 EventDispatcher 클래스가 있습니다. 다음 표현식을 사용하여 이벤트 패키지를 참조할 수 있습니다.

```
flash.events
```

이 표현식을 사용하여 EventDispatcher 클래스를 참조할 수도 있습니다.

```
flash.events.EventDispatcher
```

슬래시 구문

ActionScript 3.0에서는 슬래시 구문을 지원하지 않습니다. 슬래시 구문은 무비 클립의 경로 또는 변수를 나타내기 위해 이전 버전의 ActionScript에서 사용되었습니다.

리터럴

*리터럴*은 코드에 직접적으로 나타나는 값입니다. 다음에 나오는 예제는 모두 리터럴입니다.

```
17
"hello"
-3
9.4
null
undefined
true
false
```

리터럴을 그룹화하여 복합 리터럴을 만들 수도 있습니다. 배열 리터럴은 대괄호(`[]`)로 묶이며 쉼표를 사용하여 배열 요소를 구분합니다.

배열 리터럴은 배열을 초기화할 때 사용할 수 있습니다. 다음 예제에서는 배열 리터럴을 사용하여 초기화된 두 개의 배열을 보여 줍니다. `new` 문을 사용하여 복합 리터럴을 Array 클래스 생성자에 매개 변수로 전달할 수도 있고, Object, Array, String, Number, int, uint, XML, XMLList 및 Boolean 등의 ActionScript 기본 클래스의 인스턴스를 인스턴스화할 때 직접 리터럴 값을 지정할 수도 있습니다.

```
// new 문을 사용합니다 .
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// 리터럴을 직접 지정합니다 .
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

리터럴을 사용하여 일반 객체를 초기화할 수도 있습니다. 일반 객체는 Object 클래스의 인스턴스입니다. 객체 리터럴은 중괄호({})로 묶이며 쉼표를 사용하여 객체 속성을 구분합니다. 각 속성은 콜론(:)으로 선언되며 이는 속성 이름과 속성 값을 구분합니다.

new 문을 사용하여 일반 객체를 만들고 객체 리터럴을 Object 클래스 생성자에 매개 변수로 전달할 수도 있고, 선언할 인스턴스에 직접 객체 리터럴을 지정할 수 있습니다. 다음 예제에서는 새 일반 객체를 만든 다음 세 가지 속성 propA, propB 및 propC를 사용하여 객체를 초기화합니다. 여기서 속성 값은 각각 1, 2 및 3으로 설정합니다.

```
// new 문을 사용합니다 .
var myObject:Object = new Object({propA:1, propB:2, propC:3});

// 리터럴을 직접 지정합니다 .
var myObject:Object = {propA:1, propB:2, propC:3};
```

자세한 내용은 194페이지의 “문자열의 기초”, 270페이지의 “일반 표현식의 기초” 및 334페이지의 “XML 변수 초기화”를 참조하십시오.

세미콜론

세미콜론(;)을 사용하여 명령문을 종결할 수 있습니다. 또는 세미콜론을 생략할 수도 있으며 이 경우 컴파일러에서 각 코드 행을 단일 명령문으로 간주합니다. 많은 프로그래머가 세미콜론을 사용하여 명령문을 끝내는 것에 익숙하기 때문에 명령문을 종결할 때 세미콜론을 사용하면 보다 쉽게 코드를 읽을 수 있습니다.

세미콜론을 사용하여 명령문을 종결할 경우 한 행에 여러 개의 명령문을 배치할 수 있지만 그럴 경우 코드를 읽기가 어려워질 수 있습니다.

괄호

ActionScript 3.0에서는 괄호(())를 세 가지 방식으로 사용할 수 있습니다. 첫 번째, 괄호를 사용하여 표현식에서 연산 순서를 변경할 수 있습니다. 괄호 안에 그룹화되어 있는 연산이 항상 가장 먼저 실행됩니다. 예를 들어 다음 코드에서는 괄호를 사용하여 연산 순서를 변경합니다.

```
trace(2 + 3 * 4);     // 14
trace( (2 + 3) * 4); // 20
```

두 번째, 쉼표 연산자(,)가 포함된 괄호를 사용하여 다음 예제와 같이 일련의 표현식을 평가하고 최종 표현식의 결과를 반환합니다.

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

세 번째, 괄호를 사용하여 다음 예제와 같이 함수 또는 메서드에 하나 이상의 매개 변수를 전달할 수 있습니다. 다음 예제에서는 String 값을 `trace()` 함수에 전달합니다.

```
trace("hello"); // hello
```

주석

ActionScript 3.0 코드에 두 가지 유형의 주석 즉, 한 줄 주석 및 여러 줄 주석을 사용할 수 있습니다. 이러한 주석 처리 메커니즘은 C++ 및 Java에서의 주석 처리 메커니즘과 유사합니다. 컴파일러는 주석으로 표시된 텍스트를 무시합니다.

한 줄 주석은 두 개의 슬래시(//)로 시작하고 줄이 끝날 때까지 계속됩니다. 예를 들어 다음 코드에 한 줄 주석이 포함되어 있습니다.

```
var someNumber:Number = 3; // 한 줄 주석
```

여러 줄 주석은 슬래시와 별표(/*)로 시작하고 별표와 슬래시(*/)로 끝납니다.

```
/* This is multiline comment that can span
more than one line of code. */
```

키워드 및 예약어

*예약어*는 ActionScript에서 사용하도록 예약된 단어이기 때문에 코드에서 식별자로 사용할 수 없는 단어입니다. 예약어에는 컴파일러에 의해 프로그램 네임스페이스에서 제거된 *사전식 키워드*가 포함됩니다. 사전식 키워드를 식별자로 사용하면 컴파일러에서 오류를 보고합니다. 다음 표에 ActionScript 3.0의 사전식 키워드가 나와 있습니다.

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof

use	var	void	while
with			

식별자로 사용할 수 있지만 특정 컨텍스트에서 특별한 의미를 나타내는 *구문 키워드*라는 작은 키워드 집합이 있습니다. 다음 표에 ActionScript 3.0의 구문 키워드가 나와 있습니다.

each	get	set	namespace
include	dynamic	final	native
override	static		

*나중에 사용할 수 있는 예약어*라고도 하는 식별자가 몇 가지 있습니다. 이러한 식별자 중 일부는 ActionScript 3.0을 통합하는 소프트웨어에서 키워드로 간주될 수 있지만 ActionScript 3.0에서는 예약어가 아닙니다. 코드에 이러한 여러 식별자를 사용할 수 있지만 후속 언어 버전에서 이들이 키워드로 표시될 수 있으므로 사용하지 않는 것이 좋습니다.

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	~	transient	type
virtual	volatile		

상수

ActionScript 3.0에서는 상수를 만드는 데 사용할 수 있는 `const` 문을 지원합니다. 상수는 변경할 수 없는 고정된 값을 갖는 속성입니다. 상수에는 값을 한 번만 지정할 수 있으며 상수의 선언과 아주 근접한 위치에서 값을 지정해야 합니다. 예를 들어, 상수를 클래스의 멤버로 선언하는 경우 선언의 일부로서 해당 상수에 값을 지정하거나 클래스 생성자 내에서 값을 지정할 수 있습니다.

다음 코드에서는 두 개의 상수를 선언합니다. 첫 번째 상수 `MINIMUM`은 선언문의 일부로 지정된 값을 갖습니다. 두 번째 상수 `MAXIMUM`은 생성자에서 지정된 값을 갖습니다.

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;

  public function A()
  {
    MAXIMUM = 10;
  }
```

```
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

다른 방법으로 초기 값을 상수에 지정하려고 하면 오류가 발생합니다. 예를 들어, 클래스 외부에서 MAXIMUM의 초기 값을 설정하려고 하면 런타임 오류가 발생합니다.

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // 런타임 오류
```

Flash Player API에는 사용자를 위해 다양한 상수가 정의되어 있습니다. 규칙에 따라 ActionScript의 상수에는 밑줄(_)로 구분된 단어와 함께 모든 대문자를 사용할 수 있습니다. 예를 들어, MouseEvent 클래스 정의에서는 이 이름 지정 규칙을 사용하여 상수를 정의하며, 각 상수는 마우스 입력과 연관된 이벤트를 나타냅니다.

```
package flash.events
{
  public class MouseEvent extends Event
  {
    public static const CLICK:String               = "click";
    public static const DOUBLE_CLICK:String        = "doubleClick";
    public static const MOUSE_DOWN:String          = "mouseDown";
    public static const MOUSE_MOVE:String          = "mouseMove";
    ...
  }
}
```

연산자

연산자는 하나 이상의 피연산자를 사용하여 값을 반환하는 특수 함수입니다. 일반적으로 *피연산자*는 연산자에서 입력값으로 사용하는 리터럴, 변수 또는 표현식 등의 값입니다. 예를 들어, 다음 코드에서 더하기(+) 및 곱하기(*) 연산자는 세 개의 리터럴 피연산자(2, 3, 4)와 함께 사용되어 결과 값을 반환합니다. 이 값은 반환된 값 14를 sumNumber 변수에 지정하는 대입 연산자(=)에서 사용합니다.

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

연산자에는 단항, 이항 또는 삼항 연산자가 있습니다. *단항* 연산자는 한 개의 피연산자를 사용합니다. 예를 들어 증가 연산자(++)는 한 개의 피연산자만 사용하므로 단항 연산자입니다. *이항* 연산자는 두 개의 피연산자를 사용합니다. 예를 들어, 나누기(/) 연산자는 두 개의 피연산자를 사용합니다. *삼항* 연산자는 세 개의 피연산자를 사용합니다. 예를 들어 조건 연산자(?:)는 세 개의 피연산자를 사용합니다.

일부 연산자는 *오버로드*할 수 있습니다. 이는 연산자에 전달된 피연산자의 유형 또는 개수에 따라 연산자의 동작이 달라진다는 것을 의미합니다. 더하기(+) 연산자는 피연산자의 데이터 유형에 따라 다르게 동작하는 오버로드된 연산자의 예입니다. 피연산자가 모두 숫자인 경우 더하기 연산자는 값의 합계를 반환합니다. 피연산자가 모두 문자열이면 더하기 연산자는 두 피연산자를 연결한 문자열을 반환합니다. 다음 예제 코드에서는 연산자가 피연산자에 따라 다르게 동작하는 방식을 보여 줍니다.

```
trace(5 + 5);      // 10
trace("5" + "5"); // 55
```

연산자는 제공된 피연산자의 수에 따라 다르게 동작할 수도 있습니다. 빼기(-) 연산자는 단항 연산자이면서 이항 연산자입니다. 피연산자가 한 개만 제공되면 빼기 연산자는 피연산자를 음수로 만들어 결과를 반환합니다. 피연산자가 두 개 제공되면 빼기 연산자는 피연산자 간의 차이를 반환합니다. 다음 예제에서는 먼저 단항 연산자로 사용된 다음 이항 연산자로 사용된 빼기 연산자를 보여 줍니다.

```
trace(-3);  // -3
trace(7-2); // 5
```

연산자 우선 순위와 연산 순서

연산자 우선 순위와 연산 순서에 따라 연산자 처리 순서가 결정됩니다. 산술에 익숙한 독자에게는 컴파일러에서 더하기(+) 연산자보다 곱하기(*) 연산자를 먼저 처리하는 것이 당연해 보이겠지만 컴파일러에게는 어떤 연산자를 먼저 처리할지에 대한 명시적인 지시 사항이 필요합니다. 이러한 지시 사항을 전체적으로 *연산자 우선 순위*라고 합니다. ActionScript에 정의된 기본 연산자 우선 순위는 괄호(()) 연산자를 사용하여 변경할 수 있습니다. 예를 들어 다음 코드는 컴파일러에서 곱하기 연산자보다 더하기 연산자를 먼저 처리하도록 이전 예제의 기본 우선 순위를 변경합니다.

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

우선 순위가 동일한 둘 이상의 연산자가 동일한 표현식에 나타나는 경우가 발생할 수도 있습니다. 이러한 경우 컴파일러는 *연산 순서* 규칙을 사용하여 먼저 처리할 연산자를 결정합니다. 대입 연산자를 제외한 모든 이진 연산자는 *왼쪽 연관*이므로 왼쪽에 있는 연산자가 오른쪽에 있는 연산자보다 먼저 처리됩니다. 대입 연산자와 조건 연산자(?:)는 *오른쪽 연관*이므로 오른쪽에 있는 연산자가 왼쪽에 있는 연산자보다 먼저 처리됩니다.

예를 들어, 보다 작음(<) 연산자와 보다 큼(>) 연산자의 경우 우선 순위가 동일합니다. 두 연산자는 왼쪽 연관이므로 동일한 표현식에 사용될 경우 왼쪽에 있는 연산자가 먼저 처리됩니다. 즉, 다음 두 명령문의 결과는 동일합니다.

```
trace(3 > 2 < 1);   // false
trace((3 > 2) < 1); // false
```

보다 큼 연산자가 먼저 처리되며 피연산자 3이 피연산자 2보다 크기 때문에 `true`를 반환합니다. 그런 다음 `true` 값이 피연산자 1과 함께 보다 작음 연산자로 전달됩니다. 다음 코드는 이 중간 상태를 나타냅니다.

```
trace((true) < 1);
```

보다 작음 연산자는 `true` 값을 숫자 값 1로 변환하고 두 번째 피연산자 1과 비교한 후 값 1이 1보다 작지 않으므로 `false` 값을 반환합니다.

```
trace(1 < 1); // false
```

괄호 연산자를 사용하면 기본적으로 적용되는 왼쪽 연산 순서를 변경할 수 있습니다. 연산자와 피연산자를 괄호로 묶어 컴파일러에서 보다 작음 연산자를 먼저 처리하도록 지시할 수 있습니다. 다음 예제에서는 괄호 연산자를 사용하여 이전 예제에서와 동일한 숫자로 다른 결과를 생성합니다.

```
trace(3 > (2 < 1)); // true
```

보다 작음 연산자가 먼저 처리되며 피연산자 2가 피연산자 1보다 크기 때문에 `false` 값을 반환합니다. 그런 다음 `false` 값이 피연산자 3과 함께 보다 큼 연산자로 전달됩니다. 다음 코드는 이 중간 상태를 나타냅니다.

```
trace(3 > (false));
```

보다 큼 연산자는 false 값을 숫자 값 0으로 변환하고 다른 피연산자 3과 비교한 후 값 3이 0보다 크므로 true 값을 반환합니다.

```
trace(3 > 0); // true
```

다음 표에는 ActionScript 3.0의 연산자가 우선 순위의 내림차순으로 나열되어 있습니다. 표의 각 행에는 우선 순위가 동일한 연산자가 포함되어 있습니다. 이 표에서 각 행의 연산자는 아래 행에 있는 연산자보다 우선 순위가 높습니다.

그룹	연산자
기본	[] {x:y} () f(x) new x.y x[y] <></> @ :: ..
후위	x++ x--
단항	++x --x + - ~ ! delete typeof void
곱셈	* / %
덧셈	+ -
비트 시프트	<< >> >>>
비교	< > <= >= as in instanceof is
항등	== != === !==
비트 AND	&
비트 XOR	^
비트 OR	|
논리 AND	&&
논리 OR	||
조건	?:
대입	= *= /= %= += -= <<= >>= >>>= &= ^= |=
쉼표	,

기본 연산자

기본 연산자에는 Array 및 Object 리터럴을 만들고, 표현식을 그룹화하고, 함수를 호출하고, 클래스 인스턴스를 인스턴스화하며, 속성에 액세스하는 데 사용되는 연산자가 포함됩니다.

다음 표에 나열된 기본 연산자의 우선 순위는 모두 동일합니다. E4X 사양의 일부인 연산자는 E4X 표기법에 따라 표시합니다.

연산자	수행하는 연산
`[]`	배열 초기화
`{x:y}`	객체 초기화
`()`	표현식 그룹화
`f(x)`	함수 호출
`new`	생성자 호출
`x.y x[y]`	속성에 액세스
`<></>`	XMLList 객체 초기화(E4X)
`@`	특성에 액세스(E4X)
`::`	이름 정규화(E4X)
`..`	자손 XML 요소에 액세스(E4X)

후위 연산자

후위 연산자는 하나의 피연산자를 사용하며 값을 증가시키거나 감소시킵니다. 이 연산자는 단항 연산자이나, 높은 우선 순위와 특별한 비헤이비어 때문에 다른 단항 연산자와는 별개로 분류됩니다. 후위 연산자를 큰 표현식의 일부분으로 사용할 경우 후위 연산자가 처리되기 전에 표현식 값이 반환됩니다. 예를 들어 다음 코드에서는 값이 증가되기 전에 `xNum++` 표현식의 값이 반환되는 방식을 보여 줍니다.

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum);   // 1
```

다음 표에 나열된 후위 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
`++`	증가(후위)
`--`	감소(후위)

단항 연산자

단항 연산자는 하나의 피연산자를 사용합니다. 이 그룹의 증가(++) 및 감소(--) 연산자는 *전위 연산자*로 표현식에서 피연산자 앞에 나타납니다. 전위 연산자의 경우 전체 표현식 값이 반환되기 전에 증가 또는 감소 연산이 완료된다는 점에서 후위 연산자와 다릅니다. 예를 들어 다음 코드에서는 값이 증가한 후에 ++xNum 표현식의 값이 처리되는 방식을 보여 줍니다.

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum);   // 1
```

다음 표에 나열된 단항 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
++	증가(전위)
--	감소(전위)
+	단항 덧셈
-	단항 뺄셈(부정)
!	논리 NOT
~	비트 NOT
delete	속성 삭제
typeof	유형 정보 반환
void	정의되지 않은 값 반환

곱셈 연산자

곱셈 연산자는 피연산자 2개를 사용하여 곱하기, 나누기, 모듈러스 등을 계산합니다.

다음 표에 나열된 곱셈 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
*	곱하기
/	나누기
%	모듈러스

덧셈 연산자

덧셈 연산자는 피연산자 2개를 사용하여 더하기 또는 빼기를 계산합니다. 다음 표에 나열된 덧셈 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
+	더하기
-	빼기

비트 시프트 연산자

비트 시프트 연산자는 피연산자 2개를 사용하며 첫 번째 피연산자의 비트를 두 번째 피연산자로 지정한 비트 수 만큼 이동합니다. 다음 표에 나열된 비트 시프트 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
<<	비트 왼쪽 시프트
>>	비트 오른쪽 시프트
>>>	(비트 부호 없는 오른쪽 시프트)

비교 연산자

비교 연산자는 피연산자 2개를 사용하여 값을 비교하며 부울 값을 반환합니다.
다음 표에 나열된 비교 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
<	보다 작음
>	보다 큼
<=	보다 작거나 같음
>=	보다 크거나 같음
as	데이터 유형 확인
in	객체 속성 확인
instanceof	프로토타입 체인 확인
is	데이터 유형 확인

항등 연산자

항등 연산자는 피연산자 2개를 사용하여 값을 비교하고 부울 값을 반환합니다.
다음 표에 나열된 항등 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
==	항등
!=	비항등
===	완전 항등
!==	완전 비항등

비트 논리 연산자

비트 논리 연산자는 피연산자 2개를 사용하여 비트 수준 논리 연산을 수행합니다. 비트 논리 연산자 간에는 우선 순위가 다릅니다. 다음 표에는 비트 논리 연산자가 우선 순위의 내림차순으로 나열되어 있습니다.

연산자	수행하는 연산
&	비트 AND
^	비트 XOR
|	비트 OR

논리 연산자

논리 연산자는 피연산자 2개를 사용하여 부울 결과를 반환합니다. 논리 연산자 간에는 우선 순위가 다릅니다. 다음 표에는 논리 연산자가 우선 순위의 내림차순으로 나열되어 있습니다.

연산자	수행하는 연산
&&	논리 AND
||	논리 OR

조건 연산자

조건 연산자는 삼항 연산자로서 피연산자 3개를 사용합니다. 조건 연산자는 `if..else` 조건문을 간단히 적용하는 방법입니다.

연산자	수행하는 연산
?:	조건

대입 연산자

대입 연산자는 두 피연산자를 사용하여 한 피연산자의 값에 따라 다른 피연산자에 값을 대입합니다. 다음 표에 나열된 대입 연산자의 우선 순위는 모두 동일합니다.

연산자	수행하는 연산
=	대입
*=	곱하기 대입
/=	나누기 대입
%=	모듈러스 대입
+=	더하기 대입
-=	빼기 대입
<<=	비트 왼쪽 시프트 대입
>>=	비트 오른쪽 시프트 대입
>>>=	비트 부호 없는 오른쪽 시프트 대입
&=	비트 AND 대입
^=	비트 XOR 대입
|=	비트 OR 대입

조건문

ActionScript 3.0에서는 프로그램 흐름을 제어하는 데 사용할 수 있는 세 가지 기본 조건문을 제공합니다.

if..else

`if..else` 조건문을 사용하면 조건을 테스트한 다음 해당 조건이 존재하면 코드 블록을 실행하고 해당 조건이 존재하지 않으면 다른 코드 블록을 실행할 수 있습니다. 예를 들어 다음 코드에서는 `x` 값이 20을 초과하는지 테스트한 다음 20을 초과하면 `trace()` 함수를 호출하고 20을 초과하지 않으면 다른 `trace()` 함수를 호출합니다.

```
if (x > 20)
{
  trace("x is > 20");
}
else
{
  trace("x is <= 20");
}
```

다른 코드 블록을 실행하지 않으려면 else 문 없이 if 문을 사용할 수 있습니다.

if..else if

if..else if 조건문을 사용하여 둘 이상의 조건을 테스트할 수 있습니다. 다음 코드는 x 값이 20을 초과하는지 여부를 테스트할 뿐 아니라 x 값이 음수인지 여부도 테스트합니다.

```
if (x > 20)
{
  trace("x is > 20");
}
else if (x < 0)
{
  trace("x is negative");
}
```

if 또는 else 문 다음에 단 하나의 명령문이 오는 경우 명령문을 중괄호로 묶지 않아도 됩니다. 예를 들어 다음 코드에서는 중괄호를 사용하지 않습니다.

```
if (x > 0)
  trace("x is positive");
else if (x < 0)
  trace("x is negative");
else
  trace("x is 0");
```

그러나 명령문이 나중에 중괄호가 없는 조건문에 추가될 경우 예기치 못한 비헤이비어가 발생할 수 있으므로 항상 중괄호를 사용하는 것이 좋습니다. 예를 들어, 다음 코드에서는 조건이 true로 평가되는지에 관계없이 positiveNums의 값은 1씩 증가합니다.

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
  trace("x is positive");
  positiveNums++;

trace(positiveNums); // 1
```

switch

switch 문은 동일한 조건 표현식에 따라 실행 경로가 여러 가지로 달라지는 경우에 유용합니다. 이는 일련의 if..else if 문과 유사하지만 보다 쉽게 읽을 수 있습니다. switch 문은 조건의 부울 값을 테스트하는 대신 표현식을 평가하고 그 결과를 사용하여 실행할 코드 블록을 결정합니다. 코드 블록은 case 문으로 시작하고 break 문으로 끝납니다. 예를 들어, 다음 switch 문은 Date.getDay() 메서드에서 반환된 일수에 따라 요일을 표시합니다.

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
  case 0:
    trace("Sunday");
    break;
  case 1:
    trace("Monday");
    break;
  case 2:
    trace("Tuesday");
    break;
  case 3:
    trace("Wednesday");
    break;
  case 4:
    trace("Thursday");
    break;
  case 5:
    trace("Friday");
    break;
  case 6:
    trace("Saturday");
    break;
  default:
    trace("Out of range");
    break;
}
```

반복

반복문을 사용하면 일련의 값 또는 변수를 사용하여 특정 코드 블록을 반복해서 수행할 수 있습니다. 코드 블록은 항상 중괄호({})로 묶는 것이 좋습니다. 코드 블록에 명령문이 하나만 포함되어 있으면 중괄호를 생략해도 되지만, 조건문에서와 같이 나중에 추가된 명령문을 코드 블록에서 실수로 제외할 수 있기 때문에 중괄호를 사용하는 것이 좋습니다. 코드 블록에 포함할 명령문을 나중에 추가하고 필수 중괄호는 추가하지 않은 경우 명령문이 루프의 일부로 실행되지 않습니다.

for

for 루프를 사용하면 특정 값 범위에서 변수를 반복할 수 있습니다. for 문에 초기 값으로 설정되는 변수, 루핑 종료 시점을 결정하는 조건문, 각 루프의 변수 값을 변경하는 표현식과 같은 세 개의 표현식을 제공해야 합니다. 예를 들어, 다음 코드는 5번 반복됩니다. i 변수 값은 0에서 시작하여 4에서 끝나며 각 행에 0부터 4까지의 숫자로 출력됩니다.

```
var i:int;
for (i = 0; i < 5; i++)
{
  trace(i);
}
```

for..in

for..in 루프는 객체의 속성 또는 배열 요소를 반복합니다. 예를 들어, for..in 루프를 사용하여 일반 객체의 속성을 반복할 수 있습니다. 객체 속성은 특정한 순서로 보존되지 않으므로 임의의 순서로 나타납니다.

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
  trace(i + ": " + myObj[i]);
}
// 출력 :
// x: 20
// y: 30
```

배열 요소를 반복할 수도 있습니다.

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
  trace(myArray[i]);
}
// 출력 :
// one
```

```
// two
// three
```

객체가 동적 클래스가 아닌 사용자 정의 클래스의 인스턴스이면 객체 속성을 반복할 수 없습니다. 동적 클래스의 인스턴스라도 동적으로 추가된 속성만 반복할 수 있습니다.

for each..in

`for each..in` 루프는 컬렉션의 항목을 반복합니다. XML 또는 XMLList 객체의 태그, 객체의 속성 값 또는 배열 요소 등이 컬렉션의 항목이 될 수 있습니다. 예를 들어 다음 예제에서와 같이 `for each..in` 루프를 사용하여 일반 객체의 속성을 반복할 수 있지만, `for..in` 루프와 달리 `for each..in` 루프의 반복 변수에는 속성 이름 대신 속성 값이 포함됩니다.

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
  trace(num);
}
// 출력 :
// 20
// 30
```

다음 예제와 같이 XML 또는 XMLList 객체를 반복할 수 있습니다.

```
var myXML:XML = <users>
                   <fname>Jane</fname>
                   <fname>Susan</fname>
                   <fname>John</fname>
                </users>;

for each (var item in myXML.fname)
{
  trace(item);
}
/* 출력
Jane
Susan
John
*/
```

다음 예제와 같이 배열 요소도 반복할 수 있습니다.

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
  trace(item);
}
// 출력 :
// one
// two
// three
```

객체가 봉인된 클래스의 인스턴스인 경우 객체의 속성을 반복할 수 없습니다. 동적 클래스의 인스턴스인 경우에도 고정된 모든 속성 즉, 클래스 정의의 일부로 정의된 속성은 반복할 수 없습니다.

while

while 루프는 조건이 true일 경우 반복되는 if 문과 비슷합니다. 예를 들어, 다음 코드에서는 for 루프의 예제와 동일한 결과를 생성합니다.

```
var i:int = 0;
while (i < 5)
{
  trace(i);
  i++;
}
```

for 루프 대신 while 루프를 사용할 때 한 가지 단점은 while 루프를 사용하면 무한 루프가 쉽게 작성된다는 점입니다. for 루프 예제 코드는 카운터 변수를 증가시키는 표현식을 생략하면 컴파일되지 않지만, while 루프 예제는 해당 단계를 생략해도 컴파일됩니다. 루프에 i를 증가시키는 표현식이 없으면 무한 루프가 됩니다.

do..while

do..while 루프는 코드 블록이 한 번 이상 실행되는 while 루프입니다. 이는 코드 블록이 실행된 후 조건이 확인되기 때문입니다. 다음 코드는 조건을 충족하지 않아도 출력을 생성하는 간단한 do..while 루프 예제를 보여 줍니다.

```
var i:int = 5;
do
{
  trace(i);
  i++;
} while (i < 5);
// 출력 : 5
```

함수

*함수*는 특정 작업을 수행하는 코드 블록이며 프로그램에서 다시 사용할 수 있습니다. ActionScript 3.0에는 *메서드*와 *함수 클로저*라는 두 가지 함수가 있습니다. 함수가 정의된 컨텍스트에 따라 함수를 메서드 또는 함수 클로저라고 부릅니다. 함수를 클래스 정의의 일부로 정의하거나 객체의 인스턴스에 추가한 경우 이 함수를 메서드라고 합니다. 이와 다른 방식으로 함수를 정의한 경우 이 함수를 함수 클로저라고 합니다.

ActionScript에서 함수는 항상 매우 중요한 역할을 수행해 왔습니다. 예를 들어, ActionScript 1.0에서는 `class` 키워드가 없어 생성자 함수에 의해 "클래스"가 정의되었습니다. 언어에 `class` 키워드가 추가되었지만 언어에서 제공하는 모든 기능을 활용하려면 함수를 확실하게 이해하는 것이 여전히 중요합니다. ActionScript 함수가 C++ 또는 Java와 같은 언어에서 제공되는 함수와 유사하게 동작할 것이라고 예상하는 프로그래머에게는 ActionScript 함수를 이해하는 것이 어려운 과제일 수 있습니다. 숙련된 프로그래머에게는 기본 함수 정의 및 호출이 어렵지 않을 수 있지만 ActionScript 함수의 일부 고급 기능에 대해서는 설명이 필요합니다.

기본 함수 개념

이 단원에서는 기본적인 함수 정의 및 호출 방법에 대해 설명합니다.

함수 호출

함수 식별자 뒤에 괄호 연산자(`()`)를 사용하여 함수를 호출합니다. 함수에 전달하려는 모든 매개 변수를 괄호 연산자를 사용하여 둘러쌉니다. 예를 들어, 이 설명서 전체에서 Flash Player API의 최상위 함수인 `trace()`를 사용합니다.

```
trace("Use trace to help debug your script");
```

매개 변수 없이 함수를 호출하려면 비어 있는 괄호를 사용해야 합니다. 예를 들어, 매개 변수를 사용하지 않는 `Math.random()` 메서드를 사용하여 임의의 숫자를 생성합니다.

```
var randomNum:Number = Math.random();
```

함수 정의

ActionScript 3.0에서는 함수 명령문을 사용하거나 함수 표현식을 사용하는 두 가지 방법으로 함수를 정의합니다. 정적 또는 동적인 프로그래밍 스타일 중 선호하는 스타일에 따라 방법을 선택합니다. 정적 또는 Strict 모드의 프로그래밍을 선호하는 경우 함수 명령문을 사용하여 함수를 정의합니다. 특정한 요구에 함수 표현식이 적합하면 함수 표현식을 사용하여 함수를 정의합니다. 함수 표현식은 주로 동적 또는 Standard 모드의 프로그래밍에 사용됩니다.

함수 명령문

함수 명령문은 Strict 모드에서 함수를 정의할 때 주로 이용하는 방법입니다. 함수 명령문은 `function` 키워드로 시작하며 그 뒤에 다음이 추가됩니다.

- 함수 이름
- 괄호 안에 쉼표로 구분된 목록의 매개 변수
- 함수 본문 즉, 함수를 호출하면 실행되는 중괄호로 묶인 ActionScript 코드

예를 들어, 다음 코드에서는 매개 변수가 정의된 함수를 만든 다음 매개 변수 값으로 "`hello`" 문자열을 사용하여 함수를 호출합니다.

```
function traceParameter(aParam:String)
{
  trace(aParam);
}

traceParameter("hello"); // hello
```

함수 표현식

함수를 선언하는 두 번째 방법은 함수 리터럴 또는 익명 함수라고도 하는 함수 표현식이 포함된 대입문을 사용하는 것입니다. 이는 이전 버전의 ActionScript에서 많이 사용했던 방식으로 함수 명령문을 사용하는 방식에 비해 코드가 길어집니다.

함수 표현식이 포함된 대입문은 `var` 키워드로 시작하며 그 뒤에 다음이 추가됩니다.

- 함수 이름
- 콜론 연산자(:)
- 데이터 유형을 나타내는 `Function` 클래스
- 대입 연산자(=)
- `function` 키워드
- 괄호 안에 쉼표로 구분된 목록의 매개 변수
- 함수 본문 즉, 함수를 호출하면 실행되는 중괄호로 묶인 ActionScript 코드

예를 들어, 다음 코드는 함수 표현식을 사용하여 `traceParameter` 함수를 선언합니다.

```
var traceParameter:Function = function (aParam:String)
{
  trace(aParam);
};
traceParameter("hello"); // hello
```

함수 명령문에서와는 달리 함수 이름을 지정하지 않습니다. 함수 표현식과 함수 명령문 간의 또 다른 중요한 차이점은 함수 표현식은 명령문이 아닌 표현식이라는 점입니다. 함수 명령문과 달리 함수 표현식은 독립적으로 사용할 수 없습니다. 함수 표현식은 일반적으로 대입문과 같은 명령문의 일부로만 사용할 수 있습니다. 다음 예제에서는 배열 요소에 지정된 함수 표현식을 보여 줍니다.

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
  trace(aParam);
};
traceArray[0]("hello");
```

명령문과 표현식 간에 선택

일반적으로 특정 상황에서 표현식이 필요하지 않은 경우 함수 명령문을 사용합니다. 함수 명령문이 상대적으로 간단하며 Strict 모드와 Standard 모드에서 함수 표현식보다 일관성 있게 사용할 수 있습니다.

함수 명령문은 함수 표현식이 포함된 대입문보다 쉽게 읽을 수 있습니다. 함수 명령문은 코드를 더욱 간결하게 만들어 `var` 및 `function` 키워드를 모두 사용해야 하는 함수 표현식보다 쉽게 사용할 수 있습니다.

함수 명령문을 사용하여 선언된 메서드를 호출할 때 Strict 모드와 Standard 모드 모두에서 도트 구문을 사용할 수 있다는 점에서 함수 명령문은 두 컴파일러 모드 간의 일관성이 함수 표현식보다 낫습니다. 함수 표현식으로 선언된 메서드의 경우 도트 구문을 사용하여 호출하지 못할 수도 있습니다. 예를 들어, 다음 코드에서는 함수 표현식으로 선언된 `methodExpression()` 메서드와 함수 명령문으로 선언된 `methodStatement()` 메서드가 있는 Example 클래스를 정의합니다. Strict 모드에서는 `methodExpression()` 메서드를 호출하기 위해 도트 구문을 사용할 수 없습니다.

```
class Example
{
  var methodExpression = function() {}
  function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // Strict 모드에서는 오류가 발생하고 Standard 모드에서는
  오류가 발생하지 않습니다 .
myEx.methodStatement(); // Strict 모드와 Standard 모드에서 오류가 발생하지 않습니다 .
```

함수 표현식은 런타임 또는 동적인 비헤이비어에 중점을 두는 프로그래밍에 사용하는 것이 보다 적합합니다. Strict 모드를 사용하려고 하지만 함수 표현식으로 선언된 메서드를 호출해야 하는 경우 다음 두 가지 방법 중 하나를 사용할 수 있습니다. 첫 번째, 도트(.) 연산자 대신 대괄호([])를 사용하여 메서드를 호출할 수 있습니다. Strict 모드와 Standard 모드 모두에서 다음 메서드 호출이 성공했습니다.

```
myExample["methodLiteral"]();
```

두 번째, 전체 클래스를 동적 클래스로 선언할 수 있습니다. 이렇게 하면 도트 연산자를 사용하여 메서드를 호출할 수 있지만 해당 클래스의 모든 인스턴스에 대한 Strict 모드 기능의 일부가 제대로 작동하지 않을 수 있습니다. 예를 들어, 동적 클래스의 인스턴스의 정의되지 않은 속성에 액세스하려고 하는 경우 컴파일러에서 오류를 생성하지 않습니다.

함수 표현식을 사용하는 것이 유용한 경우도 있습니다. 함수 표현식은 한 번만 사용하고 폐기할 함수를 정의하는 경우에 일반적으로 사용됩니다. 이보다 사용 빈도는 낮지만 프로토타입 속성에 함수를 연결하는 경우에도 사용됩니다. 자세한 내용은 167페이지의 “프로토타입 객체”를 참조하십시오.

사용할 방법을 선택할 때는 함수 명령문과 함수 표현식 간의 두 가지 미세한 차이점을 고려해야 합니다. 첫 번째 차이점은 메모리 관리 및 가비지 컬렉션과 관련해서 함수 표현식은 독립된 개체가 아니라는 점입니다. 즉, 배열 요소 또는 객체 속성과 같은 다른 객체에 함수 표현식을 지정할 때는 코드에서 해당 함수 표현식에 대한 참조만 만듭니다. 함수 표현식이 연결된 배열 또는 객체가 범위를 벗어나거나 더 이상 사용할 수 없는 경우 함수 표현식에 더 이상 액세스할 수 없습니다. 배열 또는 객체를 삭제하면 함수 표현식에서 사용하는 메모리가 가비지 컬렉션의 대상이 되므로 해당 메모리를 다른 용도로 이용하거나 재사용할 수 있습니다.

다음 예제에서는 함수 표현식의 경우 표현식이 지정된 속성이 삭제되면 함수를 더 이상 사용할 수 없다는 것을 보여 줍니다. Test 클래스는 동적이므로 함수 표현식이 저장된 functionExp 속성을 추가할 수 있습니다. 도트 연산자를 사용하여 functionExp() 함수를 호출할 수 있지만 functionExp 속성이 삭제되면 더 이상 함수에 액세스할 수 없습니다.

```
dynamic class Test {}
var myTest:Test = new Test();

// 함수 표현식
myTest.functionExp = function () { trace("function expression") };
myTest.functionExp();   // 함수 표현식
delete myTest.functionExp;
myTest.functionExp();   // 오류
```

반면에 먼저 함수 명령문으로 함수를 정의하면 이 함수는 독립된 객체로 존재하며 함수가 연결된 속성을 삭제한 후에도 계속해서 사용할 수 있습니다. delete 연산자는 객체의 속성에 대해서만 작동하므로 stateFunc() 함수 자체를 삭제하도록 호출해도 작동하지 않습니다.

```
dynamic class Test {}
var myTest:Test = new Test();
```

```
// 함수 명령문
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // 함수 명령문
delete myTest.statement;
delete stateFunc;   // 영향 없음
stateFunc();        // 함수 명령문
myTest.statement(); // 오류
```

함수 명령문과 함수 표현식 간의 두 번째 차이점은 함수 명령문은 함수 명령문 앞에 나오는 명령문을 포함하여 함수 명령문이 정의된 범위 전체에서 사용된다는 점입니다. 반면에 함수 표현식은 후속 명령문에 대해서만 정의됩니다. 예를 들어 다음 코드에서는 `scopeTest()` 함수를 정의하기 전에 해당 함수를 성공적으로 호출합니다.

```
statementTest(); // statementTest

function statementTest():void
{
  trace("statementTest");
}
```

함수 표현식은 정의하기 전에 사용할 수 없기 때문에 다음 코드에서 런타임 오류가 발생합니다.

```
expressionTest(); // 런타임 오류

var expressionTest:Function = function ()
{
  trace("expressionTest");
}
```

함수에서 값 반환

함수에서 값을 반환하려면 `return` 문 다음에 반환할 표현식 또는 리터럴 값을 지정합니다. 예를 들어, 다음 코드는 매개 변수를 나타내는 표현식을 반환합니다.

```
function doubleNum(baseNum:int):int
{
  return (baseNum * 2);
}
```

`return` 문은 함수를 종료하므로 다음과 같이 `return` 문 아래의 명령문이 모두 실행되지 않습니다.

```
function doubleNum(baseNum:int):int {
  return (baseNum * 2);
  trace("after return"); // 이 trace 문은 실행되지 않습니다 .
}
```

반환 유형을 지정하기로 결정한 경우 Strict 모드에서는 적절한 유형의 값을 반환해야 합니다. 예를 들어 다음 코드에서는 올바른 값을 반환하지 않기 때문에 Strict 모드에서 오류가 발생합니다.

```
function doubleNum(baseNum:int):int
{
  trace("after return");
}
```

중첩 함수

함수를 중첩할 수 있으므로 다른 함수 내에 함수를 선언할 수 있습니다. 함수에 대한 참조가 외부 코드로 전달되지 않으면 중첩 함수는 부모 함수 내에서만 사용할 수 있습니다. 예를 들어, 다음 코드는 `getNameAndVersion()` 함수 내에 두 개의 중첩 함수를 선언합니다.

```
function getNameAndVersion():String
{
  function getVersion():String
  {
    return "9";
  }
  function getProductName():String
  {
    return "Flash Player";
  }
  return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 9
```

중첩 함수가 외부 코드로 전달되면 해당 함수는 함수 클로저로 전달되므로 함수가 정의될 때 범위에 있는 모든 정의가 함수에 그대로 유지됩니다. 자세한 내용은 129페이지의 "함수 클로저"를 참조하십시오.

함수 매개 변수

ActionScript 3.0의 함수 매개 변수에는 이 언어를 처음 접하는 프로그래머에게는 신기할 수 있는 몇 가지 기능이 있습니다. 대부분의 프로그래머는 값 또는 참조에 의한 매개 변수 전달이라는 개념에 익숙하지만 `arguments` 객체 및 ...(rest) 매개 변수에는 익숙하지 않을 수 있습니다.

값 또는 참조에 의한 인수 전달

여러 프로그래밍 언어에서 값에 의한 인수 전달과 참조에 의한 인수 전달 간의 차이점은 코드를 설계하는 방법에 영향을 미칠 수 있기 때문에 잘 이해해야 합니다.

값에 의한 전달은 인수 값이 함수 내에서 사용할 로컬 변수에 복사되는 것을 의미합니다. 참조에 의한 전달은 실제 값 대신 인수에 대한 참조만 전달되는 것을 의미합니다. 이 경우 실제 인수가 복사되지는 않습니다. 대신 인수로 전달된 변수에 대한 참조가 만들어져 함수 내에서 사용할 로컬 변수에 지정됩니다. 로컬 변수를 함수 외부의 변수에 대한 참조로 사용하여 원본 변수의 값을 변경할 수 있습니다.

ActionScript 3.0에서는 모든 값이 객체로 저장되므로 모든 인수가 참조를 기준으로 전달됩니다. 그러나 Boolean, Number, int, uint 및 String 등이 포함된 프리미티브 데이터 유형에 속한 객체에는 값을 기준으로 전달된 것과 같이 동작하도록 하는 특수 연산자가 있습니다. 예를 들어, 다음 코드에서는 xParam 및 yParam이라고 하는 int 유형의 두 매개 변수를 정의하는 passPrimitives() 함수를 만듭니다. 이러한 매개 변수는 passPrimitives() 함수 본문 내에 선언된 로컬 변수와 유사합니다. xValue 및 yValue 인수를 사용하여 함수를 호출하면 xParam 및 yParam 매개 변수는 xValue 및 yValue로 표시되는 int 객체에 대한 참조로 초기화됩니다. 인수가 프리미티브 값이기 때문에 값을 기준으로 전달된 것처럼 동작합니다. 처음에는 xParam 및 yParam에 xValue 및 yValue 객체에 대한 참조만 포함되지만 함수 본문 내에서 xParam 및 yParam의 값을 변경하면 메모리에 새 값의 복사본이 생성됩니다.

```
function passPrimitives(xParam:int, yParam:int):void
{
  xParam++;
  yParam++;
  trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);          // 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);          // 10 15
```

passPrimitives() 함수 내에서는 xParam 및 yParam의 값이 증가되지만 마지막 trace 문에서와 같이 xValue 및 yValue의 값에는 영향을 주지 않습니다. 함수 내에 있는 xValue 및 yValue는 함수 외부에 있는 동일한 이름의 변수와 별개로 존재하는 메모리의 새 위치를 가리키기 때문에 매개 변수가 xValue 및 yValue 변수와 이름이 동일한 경우에도 영향을 미치지 않습니다.

프리미티브 데이터 유형에 속하지 않은 다른 모든 객체는 항상 참조로 전달되므로 원본 변수의 값을 변경할 수 있습니다. 예를 들어, 다음 코드에서는 x와 y라는 두 가지 속성이 있는 objVar 객체를 만듭니다. 이 객체는 passByRef() 함수에 인수로 전달됩니다. 이 객체는 프리미티브 유형이 아니기 때문에 참조를 기준으로 전달될 뿐만 아니라 참조를 유지합니다. 즉, 함수 내에 있는 매개 변수가 변경되면 함수 외부의 객체 속성에도 적용됩니다.

```
function passByRef(objParam:Object):void
{
  objParam.x++;
  objParam.y++;
  trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar);         // 11 16
trace(objVar.x, objVar.y); // 11 16
```

objParam 매개 변수가 objVar 전역 변수와 동일한 객체를 참조합니다. 예제의 trace 문에서 볼 수 있듯이 objParam 객체의 x 및 y 속성을 변경하면 objVar 객체에 반영됩니다.

매개 변수 기본값

ActionScript 3.0에는 함수의 *매개 변수 기본값*을 선언할 수 있는 새로운 기능이 있습니다. 매개 변수 기본값이 있는 함수를 호출할 때 기본값이 있는 매개 변수를 생략하면 해당 매개 변수에 대해 함수 정의에 지정된 값이 사용됩니다. 기본값이 있는 모든 매개 변수는 매개 변수 목록의 끝에 배치해야 합니다. 기본값으로 지정된 값은 컴파일 타임 상수여야 합니다. 매개 변수에 대해 기본값을 사용하여 매개 변수를 효과적으로 *선택적 매개 변수*로 만들 수 있습니다. 기본값이 없는 매개 변수는 *필수 매개 변수*로 간주됩니다.

예를 들어, 다음 코드는 세 개의 매개 변수가 있는 함수를 만들며 이 중 두 매개 변수에는 기본값이 있습니다. 하나의 매개 변수만 사용하여 함수를 호출하면 나머지 매개 변수에는 기본값이 사용됩니다.

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
  trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

arguments 객체

함수에 인수가 전달되면 arguments 객체를 사용하여 함수에 전달된 매개 변수에 대한 정보에 액세스할 수 있습니다. arguments 객체는 다음과 같은 측면에서 고찰해야 합니다.

- arguments 객체는 함수에 전달된 매개 변수를 모두 포함하는 배열입니다.
- arguments.length 속성에는 함수에 전달된 매개 변수의 수가 저장됩니다.
- arguments.callee 속성은 함수 자체에 대한 참조를 제공하며 이는 함수 표현식에 대한 재귀 호출에 유용합니다.

예외

arguments 객체를 사용할 수 없는 경우, 이름이 arguments인 매개 변수가 있거나 사용자가 ...(rest) 매개 변수를 사용하는 경우

ActionScript 3.0에서는 함수 정의에 정의된 매개 변수보다 많은 매개 변수를 사용하여 함수를 호출할 수 있지만, 매개 변수의 수가 필수 매개 변수의 수보다 적으면 Strict 모드에서 컴파일러 오류가 발생합니다. arguments 객체를 배열로 사용하면 해당 매개 변수가 함수 정의에 정의되어 있는지 여부에 관계없이 함수에 전달된 모든 매개 변수에 액세스할 수 있습니다. 다음 예제에서는 arguments.length 속성과 함께 arguments 배열을 사용하여 traceArgArray() 함수에 전달된 매개 변수를 모두 추적합니다.

```
function traceArgArray(x:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
```

```
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);

// 출력 :
// 1
// 2
// 3
```

arguments.callee 속성은 주로 재귀를 만들 때 익명 함수에 사용됩니다. 이 속성을 사용하여 코드에 융통성을 추가할 수 있습니다. 함수 이름 대신 arguments.callee를 사용하면 개발 주기 과정에서 재귀 함수 이름이 변경된 경우 함수 본문에서 재귀 호출을 변경하는 것에 신경 쓰지 않아도 됩니다. arguments.callee 속성은 다음 함수 표현식에서 재귀를 활성화하는 데 사용됩니다.

```
var factorial:Function = function (x:uint)
{
  if(x == 0)
  {
    return 1;
  }
  else
  {
    return (x * arguments.callee(x - 1));
  }
}

trace(factorial(5)); // 120
```

함수 선언에서 ...(rest) 매개 변수를 사용하면 arguments 객체는 사용할 수 없습니다. 대신 선언한 매개 변수 이름을 사용하여 매개 변수에 액세스해야 합니다.

또한 매개 변수 이름으로 "arguments" 문자열을 사용하면 arguments 객체가 가려지므로 이렇게 사용하지 않는 것이 좋습니다. 예를 들어, traceArgArray() 함수가 다시 작성되어 arguments 매개 변수가 추가되는 경우 함수 본문에 있는 arguments에 대한 참조를 사용하면 arguments 객체가 아닌 매개 변수를 참조하게 됩니다. 다음 코드는 출력을 생성하지 않습니다.

```
function traceArgArray(x:int, arguments:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);
```

```
// 출력되는 내용이 없습니다 .
```

이전 버전의 ActionScript에 있는 arguments 객체에는 현재 함수를 호출한 함수에 대한 참조인 caller 속성도 포함되어 있습니다. ActionScript 3.0에는 caller 속성이 없지만 호출 함수에 대한 참조가 필요한 경우 호출 함수 자체에 대한 참조인 추가 매개 변수가 전달되도록 호출 함수를 변경할 수 있습니다.

...(rest) 매개 변수

ActionScript 3.0에서 ...(rest) 매개 변수라는 새 매개 변수 선언을 제공합니다. 이 매개 변수를 사용하면 쉼표로 구분된 인수를 무제한으로 받아들이는 배열 매개 변수를 지정할 수 있습니다. 매개 변수에는 예약어를 제외한 모든 이름이 포함될 수 있습니다. 이 매개 변수 선언은 마지막으로 지정된 매개 변수여야 합니다. 이 매개 변수를 사용하면 arguments 객체를 사용할 수 없습니다. ... (rest) 매개 변수에서 arguments 배열 및 arguments.length 속성과 동일한 기능을 제공하지만 arguments.callee에서 제공하는 것과 유사한 기능을 제공하지는 않습니다. ...(rest) 매개 변수를 사용하기 전에 arguments.callee를 사용할 필요가 없는지 확인해야 합니다.

다음 예제에서는 arguments 객체 대신 ...(rest) 매개 변수를 사용하여 traceArgArray() 함수를 다시 작성합니다.

```
function traceArgArray(... args):void
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// 출력 :
// 1
// 2
// 3
```

... (rest) 매개 변수가 마지막으로 나열되기만 하면 다른 매개 변수와 함께 사용할 수도 있습니다. 다음 예제에서는 int 유형의 x를 첫 번째 매개 변수로 사용하고 ... (rest) 매개 변수를 두 번째 매개 변수로 사용할 수 있도록 traceArgArray() 함수를 수정합니다. 첫 번째 매개 변수가 더 이상 ...(rest) 매개 변수에서 만든 배열의 일부가 아니기 때문에 첫 번째 값은 출력에서 제외됩니다.

```
function traceArgArray(x: int, ... args)
{
  for (var i:uint = 0; i < args.length; i++)
  {
```

```
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// 출력 :
// 2
// 3
```

객체와 같은 함수

ActionScript 3.0에서 함수는 객체입니다. 함수를 만드는 것은 객체를 만드는 것을 의미합니다. 이 객체는 다른 함수에 매개 변수로 전달될 수 있을 뿐만 아니라 다른 함수에 연결된 속성 및 메서드를 가질 수도 있습니다.

다른 함수에 인수로 전달된 함수는 값이 아닌 참조를 기준으로 전달됩니다. 함수를 인수로 전달하는 경우 식별자만 사용하며 메서드를 호출하는 데 사용하는 괄호 연산자는 사용하지 않습니다. 예를 들어, 다음 코드는 `clickListener()` 함수를 `addEventListener()` 메서드에 인수로 전달합니다.

```
addEventListener(MouseEvent.CLICK, clickListener);
```

`Array.sort()` 메서드에는 함수를 받는 매개 변수도 정의되어 있습니다. `Array.sort()` 함수에 인수로 사용된 사용자 정의 정렬 함수에 대한 예제는 217페이지의 “배열 정렬”을 참조하십시오.

ActionScript를 처음 접하는 프로그래머에게는 이상하게 보일 수 있지만 함수는 다른 모든 객체와 같이 속성과 메서드를 가질 수 있습니다. 실제로 모든 함수에는 함수에 대해 정의된 매개 변수의 수를 저장하는 `length`라는 읽기 전용 속성이 있습니다. 이 속성은 함수에 전달된 인수의 수를 저장하는 `arguments.length` 속성과는 다릅니다. ActionScript에서는 함수에 전달된 인수의 수가 해당 함수에 대해 정의된 매개 변수의 수를 초과할 수 있음을 상기하십시오. 다음 예제에서 두 속성 간의 차이점을 보여 줍니다. 이 예제의 경우 Strict 모드에서는 전달된 인수의 수와 정의된 매개 변수의 수가 정확하게 일치해야 하므로 Standard 모드에서만 컴파일됩니다.

```
function traceLength(x:uint, y:uint):void
{
  trace("전달된 인수 : " + arguments.length);
  trace("전달되어야 할 인수 : " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* 출력 :
arguments received: 4
arguments expected: 2 */
```

함수 본문 외부에 함수 속성을 정의하는 방법으로 함수 속성을 직접 정의할 수 있습니다. 함수 속성은 함수와 연관된 변수의 상태를 저장할 수 있는 준 정적 속성 역할을 수행할 수 있습니다. 예를 들어, 특정 함수가 호출된 횟수를 추적할 수 있습니다. 사용자의 특정 명령 사용 횟수를 추적하는 기능이 있는 게임을 작성하는 경우 정적 클래스 속성을 사용할 수도 있지만 함수 호출 횟수를 추적하는 기능이 유용할 수 있습니다. 다음 코드는 함수 선언 외부에 함수 속성을 만들고 함수가 호출될 때마다 속성 값을 증가시킵니다.

```
someFunction.counter = 0;

function someFunction():void
{
  someFunction.counter++;
}

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

함수 범위

함수 범위는 함수를 호출할 수 있는 프로그램의 위치뿐만 아니라 함수가 액세스할 수 있는 정의를 결정합니다. 변수 식별자에 적용되는 동일한 범위 규칙이 함수 식별자에 적용됩니다. 전역 범위에 선언된 함수는 코드 전체에 사용할 수 있습니다. 예를 들어, ActionScript 3.0에는 코드 전체에 사용할 수 있는 `isNaN()` 및 `parseInt()` 등의 전역 함수가 포함되어 있습니다. 다른 함수 내에 선언된 중첩 함수는 선언된 함수 내의 어느 곳에서나 사용할 수 있습니다.

범위 체인

함수가 실행되면 여러 객체와 속성이 만들어집니다. 첫 번째, 매개 변수 및 함수 본문에 선언된 함수 또는 로컬 변수를 저장하는 *activation 객체*라는 특수 객체가 만들어집니다. activation 객체는 내부 메커니즘이므로 직접 액세스할 수 없습니다. 두 번째, Flash Player에서 식별자 선언을 찾을 때 확인하는 정렬된 객체 목록이 포함되어 있는 *범위 체인*이 만들어집니다. 실행되는 모든 함수의 내부 속성에 범위 체인이 저장됩니다. 중첩 함수의 범위 체인은 중첩 함수의 activation 객체에서 시작하여 부모 함수의 activation 객체로 이어집니다. 전역 객체에 도달할 때까지 이러한 방식으로 체인이 계속됩니다. ActionScript 프로그램이 시작되고 모든 전역 변수 및 함수가 포함되면 전역 객체가 만들어집니다.

함수 클로저

*함수 클로저*는 함수의 스냅샷 및 *사전적 환경*이 포함된 객체입니다. 함수의 사전적 환경에는 함수 범위 체인에 있는 모든 변수, 속성, 메서드 및 객체가 해당 값과 함께 포함됩니다. 함수 클로저는 객체 또는 클래스와는 별도로 함수가 실행될 때 만들어집니다. 자신이 정의된 범위를 유지하는 함수 클로저로 인해 함수가 인수 또는 반환 값으로 다른 범위로 전달될 때 흥미로운 결과가 나타납니다.

예를 들어, 다음 코드에서는 두 개의 함수를 만듭니다. `foo()`는 사각형 면적을 계산하는 `rectArea()`라는 중첩된 함수를 반환하고 `bar()`는 `foo()`를 호출하고 반환된 함수 클로저를 `myProduct` 변수에 저장합니다. `bar()` 함수에서 로컬 변수 `x`의 값을 2로 정의하지만 함수 클로저 `myProduct()`를 호출하면 함수 클로저에서는 `foo()` 함수에 정의된 변수 `x`의 값 40이 그대로 유지됩니다. 따라서 `bar()` 함수에서 8 대신 160을 반환합니다.

```
function foo():Function
{
  var x:int = 40;
  function rectArea(y:int):int // 함수 클로저 정의됨
  {
    return x * y
  }
  return rectArea;
}
function bar():void
{
  var x:int = 2;
  var y:int = 4;
  var myProduct:Function = foo();
  trace(myProduct(4)); // 함수 클로저 호출됨
}
bar(); // 160
```

메서드 역시 자신이 만들어진 사전적 환경에 대한 정보를 유지한다는 점에서 함수 클로저와 유사하게 동작합니다. 이 특성은 인스턴스에서 메서드를 추출하여 바인딩된 메서드가 만들어질 때 가장 잘 나타납니다. 함수 클로저와 바인딩된 메서드 간의 주요 차이점은 바인딩된 메서드에 있는 `this` 키워드 값은 항상 처음에 연결된 인스턴스를 참조하는 반면 함수 클로저에 있는 `this` 키워드 값은 변경할 수 있다는 것입니다. 자세한 내용은 146페이지의 "바인딩된 메서드"를 참조하십시오.

제 4 장

ActionScript의 객체 지향 프로그래밍

4

이 장에서는 객체 지향 프로그래밍(OOP)을 지원하는 ActionScript의 요소에 대해 설명합니다. 객체 설계, 추상화, 캡슐화, 상속 및 다형성 등의 일반적인 OOP 원리에 대한 부분은 다루지 않고 ActionScript 3.0을 사용하여 이러한 원리를 적용하는 방법을 중심으로 설명합니다.

ActionScript는 원래 스크립트 언어이므로 ActionScript 3.0 OOP는 선택적으로 지원됩니다. 프로그래머는 이를 통해 프로젝트의 범위와 복잡도에 따라 가장 효과적인 방법을 선택할 수 있습니다. 소규모 작업에서는 ActionScript를 절차식 프로그래밍 패러다임으로 사용하는 것으로 충분하며 대규모 프로젝트의 경우에는 OOP 원리를 적용하여 코드를 읽기 쉽고 유지 관리 및 확장이 편리하게 만들 수 있습니다.

목차

객체 지향 프로그래밍의 기초

객체 지향 프로그래밍 소개

객체 지향 프로그래밍(OOP)은 프로그램 내의 코드를 정보(데이터 값)와 기능이 포함된 개별 요소인 객체로 그룹화하여 코드를 구성하는 방법입니다. 객체 지향 접근을 사용하여 프로그램을 구성하면 특정 정보(예: 앨범 제목, 트랙 제목 또는 아티스트 이름 등의 음악 정보)를 공통 기능 또는 해당 정보와 관련된 액션(예: "재생 목록에 트랙 추가" 또는 "해당 아티스트의 모든 노래 재생")으로 그룹화할 수 있습니다. 이와 같은 항목은 단일 항목인 객체(예: "Album" 또는 "MusicTrack")로 결합됩니다. 이러한 값과 기능을 함께 묶으면 여러 개의 변수를 추적할 필요 없이 단일 변수만 추적하거나, 관련 기능을 함께 구성하고, 실제와 보다 근접한 구조를 갖는 프로그램을 개발하는 등 몇 가지 장점이 있습니다.

일반적인 객체 지향 프로그래밍 작업

실제로 객체 지향 프로그래밍은 두 부분으로 구성됩니다. 첫 번째는 프로그램 설계에 필요한 전략 및 기술로, *객체 지향 설계*라고도 합니다. 이 내용은 광범위한 주제이기 때문에 이 장에서는 다루지 않습니다. OOP를 구성하는 또 다른 부분은 주어진 프로그래밍 언어에서 객체 지향 접근을 사용하여 프로그램을 구축하는 데 이용할 수 있는 실제 프로그래밍 구조입니다. 이 장에서는 아래와 같은 OOP의 일반적인 작업을 설명합니다.

- 클래스 정의
- 속성, 메서드, get 및 set 접근자(접근자 메서드) 만들기
- 클래스, 속성, 메서드 및 접근자에 대한 액세스 제어
- 정적 속성 및 메서드 만들기
- 열거형 구조 만들기
- 인터페이스 정의 및 사용
- 상속 항목 작업(클래스 요소 재정의 포함)

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 특성: 클래스 정의에서 속성 또는 메서드와 같은 클래스 요소에 지정된 특징입니다. 특성은 프로그램의 다른 부분에서 코드가 속성 또는 메서드에 액세스할 수 있는지 여부를 정의하는 데 일반적으로 사용됩니다. 예를 들어, `private`과 `public`은 특성입니다. 전용 메서드는 클래스 내의 코드에 의해서만 호출이 가능한 반면, 공용 메서드는 프로그램의 모든 코드에서 호출할 수 있습니다.

- 클래스: 특정 객체 유형(예: 특정 데이터 유형을 갖는 객체의 템플릿 또는 청사진)의 구조 및 비헤이비어에 대한 정의입니다.
- 클래스 계층 구조: 서로 연관된 여러 클래스의 구조를 나타내며, 어떤 클래스가 다른 클래스로부터 기능을 상속받을지를 지정합니다.
- 생성자: 클래스에서 정의할 수 있는 특수 메서드로, 클래스 인스턴스를 만들 때 호출됩니다. 생성자는 일반적으로 기본값을 지정하거나 객체 설정 작업을 수행할 때 사용됩니다.
- 데이터 유형: 특정 변수에 저장할 수 있는 정보 유형입니다. 일반적으로 *데이터 유형*은 *클래스*를 의미합니다.
- 도트 연산자: ActionScript(및 다수의 기타 프로그래밍 언어)에서 객체의 자식 요소(예: 속성, 메서드)를 지칭하기 위해 사용되는 마침표 기호(.)입니다. 예를 들어, 표현식 `myObject.myProperty`에서 도트 연산자는 `myProperty`라는 용어가 `myObject`라는 이름의 객체 요소인 일부 값을 지칭함을 나타냅니다.
- 열거형: 편의를 위해 단일 클래스의 속성으로 그룹화된 상수 값 집합입니다.
- 상속: 한 클래스 정의에 다른 클래스 정의의 모든 기능이 포함되도록 하는(일반적으로 해당 기능에 추가) OOP 메커니즘입니다.
- 인스턴스: 프로그램에서 만들어진 실제 객체입니다.
- 네임스페이스: 본질적으로 사용자 정의 특성을 뜻하며, 다른 코드에 대한 액세스 관계를 보다 세부적으로 제어할 수 있습니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장에 나오는 코드 샘플은 주로 데이터 유형 정의와 조작을 다루므로, 예제를 테스트하려면 정의된 클래스의 인스턴스를 만들고, 그 속성이나 메서드를 사용하여 인스턴스를 조작한 다음, 인스턴스의 속성 값을 확인해야 합니다. 이러한 값을 보려면 스테이지의 텍스트 필드 인스턴스에 값을 기록하거나 `trace()` 함수를 사용하여 [출력] 패널에 값을 출력합니다. 이러한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

클래스

클래스는 객체를 추상적으로 나타낸 것입니다. 클래스에는 객체에 담을 수 있는 데이터 유형 및 객체에 적용할 수 있는 비헤이비어에 대한 정보가 저장됩니다. 상호 작용하는 객체의 수가 적은 소규모 스크립트를 작성할 때는 이러한 추상화의 유용성이 잘 드러나지 않습니다. 그러나 프로그램의 범위가 커지면서 관리해야 하는 객체 수가 증가함에 따라 클래스를 통해 객체가 만들어지는 방식 및 객체가 상호 작용하는 방식을 보다 효율적으로 제어할 수 있음을 발견할 수도 있습니다.

ActionScript 1.0부터 ActionScript 프로그래머는 Function 객체를 사용하여 클래스와 유사한 구문을 만들 수 있었습니다. ActionScript 2.0에서는 `class` 및 `extends` 등의 키워드를 통해 클래스가 공식적으로 지원되었습니다. ActionScript 3.0에서는 ActionScript 2.0에 도입된 키워드가 계속 지원될 뿐만 아니라 새로운 기능이 추가되었습니다. 이러한 기능에는 `protected` 및 `internal` 특성을 통한 향상된 액세스 제어 및 `final` 및 `override` 키워드를 통한 구체적인 상속 제어 등이 있습니다.

Java, C++ 또는 C# 등의 프로그래밍 언어로 클래스를 만든 경험이 있는 경우 ActionScript에서도 익숙한 환경을 경험할 수 있습니다. ActionScript에서는 `class`, `extends` 및 `public` 등 동일한 키워드 및 특성 이름을 대부분 공유합니다. 다음 단원에서는 이러한 키워드 및 특성 이름에 대해 설명합니다.

참고 **이 장에서 *속성*이라는 용어는 변수, 상수 및 메서드를 비롯하여 객체 또는 클래스의 멤버를 나타냅니다. 또한 *클래스*와 *정적*이라는 용어가 서로 구별되지 않고 사용되는 경우가 많지만 이 장에서는 이러한 용어가 서로 구별됩니다. 예를 들어 이 장에서 *클래스 속성*이라는 용어는 정적 멤버뿐만 아니라 클래스의 모든 멤버를 나타냅니다.**

클래스 정의

ActionScript 3.0 클래스 정의에는 ActionScript 2.0 클래스 정의와 비슷한 구문이 사용됩니다. 올바른 클래스 정의 구문에는 `class` 키워드 뒤에 클래스 이름이 나옵니다. 클래스 이름 뒤에는 중괄호(`{}`)로 묶인 클래스 본문이 나옵니다. 예를 들어, 다음 코드에서는 `visible`이라는 변수 하나가 들어 있는 Shape라는 클래스가 만들어집니다.

```
public class Shape
{
  var visible:Boolean = true;
}
```

중요한 구문 변경 내용 중 하나는 패키지 안에 있는 클래스의 정의와 관련된 것입니다. ActionScript 2.0에서는 클래스가 패키지 안에 있는 경우 클래스 선언에 패키지 이름이 포함되어야 합니다. ActionScript 3.0에서는 `package` 문이 도입되어, 패키지 이름이 클래스 선언이 아닌 패키지 선언에 포함되어야 합니다. 예를 들어, 다음 클래스 선언에서는 flash.display 패키지에 속한 BitmapData 클래스가 ActionScript 2.0과 ActionScript 3.0에서 정의된 방식을 볼 수 있습니다.

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
  public class BitmapData {}
}
```

클래스 특성

ActionScript 3.0에서는 다음 네 가지 특성 중 하나를 사용하여 클래스 정의를 수정할 수 있습니다.

특성	정의
`dynamic`	런타임에 속성을 인스턴스에 추가할 수 있게 합니다.
`final`	다른 클래스에서 확장할 수 없습니다.
`internal`(기본값)	현재 패키지 내에서 참조할 수 있습니다.
`public`	모든 위치에서 참조할 수 있습니다.

`internal`의 경우를 제외하고 이러한 각 특성에 연결된 비헤이비어를 사용하려면 특성을 명시적으로 포함해야 합니다. 예를 들어, 클래스를 정의할 때 `dynamic` 특성을 포함하지 않으면 런타임 시 속성을 클래스 인스턴스에 추가할 수 없습니다. 특성을 명시적으로 지정하려면 다음 코드와 같이 클래스 정의 시작 부분에 특성을 배치합니다.

```
dynamic class Shape {}
```

`abstract`라는 특성은 이 목록에 포함되어 있지 않습니다. 이는 ActionScript 3.0에서 추상 클래스가 지원되지 않기 때문입니다. 또한 `private` 및 `protected`라는 특성도 목록에 포함되어 있지 않습니다. 이러한 특성은 클래스 정의 내에서만 의미가 있으며 클래스 자체에는 적용할 수 없습니다. 패키지 외부에서 클래스를 공용으로 참조할 수 없게 하려면 클래스를 패키지 내에 배치하고 클래스에 `internal` 특성을 지정합니다. 또는 `internal`과 `public` 특성을 모두 생략하여 컴파일러에서 자동으로 `internal` 특성을 추가하도록 합니다. 클래스가 정의된 소스 파일 외부에서 클래스를 참조할 수 없게 하려면 클래스를 소스 파일 맨 아래에서 패키지 정의의 닫는 중괄호 밑에 배치합니다.

클래스 본문

중괄호로 묶인 클래스 본문은 클래스의 변수, 상수 및 메서드를 정의하는 데 사용됩니다. 다음 예제에서는 Adobe Flash Player API의 Accessibility 클래스 선언을 보여 줍니다.

```
public final class Accessibility
{
  public static function get active():Boolean;
  public static function updateProperties():void;
}
```

클래스 본문 내에 네임스페이스를 정의할 수도 있습니다. 다음 예제에서는 클래스 본문 내에 네임스페이스를 정의하여 해당 클래스에서 메서드의 특성으로 사용하는 방법을 보여 줍니다.

```
public class SampleClass
{
  public namespace sampleNamespace;
```

```
  sampleNamespace function doSomething():void;
}
```

ActionScript 3.0에서는 클래스 본문 내에 정의뿐만 아니라 명령문도 포함시킬 수 있습니다. 클래스 본문 내에 있지만 메서드 정의 외부에 있는 명령문은 클래스 정의가 처음 실행되어 관련 클래스 객체가 만들어질 때 정확히 한 번만 실행됩니다. 다음 예제에는 외부 함수인 `hello()`에 대한 호출과 클래스가 정의될 때 확인 메시지를 출력하는 `trace` 문이 포함되어 있습니다.

```
function hello():String
{
  trace("hola");
}
class SampleClass
{
  hello();
  trace("class created");
}
// 클래스가 생성될 때 출력합니다 .
hola
class created
```

이전 버전의 ActionScript와 달리 ActionScript 3.0에서는 같은 클래스 본문 내에서 이름이 같은 정적 속성과 인스턴스 속성을 정의할 수 있습니다. 예를 들어, 다음 코드에서는 `message`라는 정적 변수 및 이름이 같은 인스턴스 변수를 선언합니다.

```
class StaticTest
{
  static var message:String = "static variable";
  var message:String = "instance variable";
}
// 스크립트에서
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // 출력 : static variable
trace(myST.message);       // 출력 : instance variable
```

클래스 속성 특성

ActionScript 객체 모델에서 *속성*이라는 용어는 변수, 상수 및 메서드 등 클래스의 멤버가 될 수 있는 모든 항목을 나타냅니다. 이 용어는 해당 용어를 보다 좁은 의미로 사용하여 변수이거나 getter 또는 setter 메서드로 정의되는 클래스 멤버만 나타내는 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 경우와 다른 방식으로 사용됩니다. ActionScript 3.0에는 클래스의 모든 속성과 함께 사용할 수 있는 특성 집합이 있습니다. 다음 표에 이러한 특성들이 나열되어 있습니다.

특성	정의
`internal`(기본값)	같은 패키지 내에서 참조할 수 있습니다.
`private`	같은 클래스에서 참조할 수 있습니다.
`protected`	같은 클래스 및 파생 클래스에서 참조할 수 있습니다.
`public`	모든 위치에서 참조할 수 있습니다.
`static`	속성이 클래스의 인스턴스가 아니라 클래스에 속하도록 지정합니다.
`UserDefinedNamespace`	사용자 정의 네임스페이스 이름입니다.

액세스 제어 네임스페이스 특성

ActionScript 3.0에서는 클래스 내에 정의된 속성에 대한 액세스를 제어하는 네 가지 특수 특성인 `public`, `private`, `protected` 및 `internal`이 제공됩니다.

`public` 특성을 사용하면 스크립트의 모든 위치에서 속성을 참조할 수 있습니다. 예를 들어, 해당 패키지 외부의 코드에서 메서드를 사용할 수 있게 하려면 메서드를 `public` 특성으로 선언해야 합니다. 이는 `var`, `const` 또는 `function` 등 속성을 정의하는 데 사용된 키워드에 관계없이 모든 속성에서 마찬가지입니다.

`private` 특성을 사용하면 해당 속성이 정의된 클래스 내에서만 속성에 액세스할 수 있습니다. 이 `private` 특성의 비헤이비어는 ActionScript 2.0에서 하위 클래스가 수퍼 클래스의 전용 속성에 액세스하도록 허용했던 비헤이비어와는 다릅니다. 또한 런타임 액세스 비헤이비어에도 중요한 차이점이 있습니다. ActionScript 2.0에서는 `private` 키워드를 사용하면 컴파일 타임에만 액세스가 금지되었고 런타임에는 이러한 제한을 쉽게 피할 수 있었습니다. ActionScript 3.0에서는 더 이상 그렇지 않습니다. `private`으로 표시된 속성은 컴파일 타임과 런타임 시 모두 사용할 수 없습니다.

예를 들어, 다음 코드에서는 전용 변수 하나가 있는 PrivateExample이라는 간단한 클래스를 만든 다음 클래스 외부에서 전용 변수에 액세스합니다. ActionScript 2.0의 경우 컴파일 타임 액세스는 금지되지만, 컴파일 타임이 아닌 런타임에 속성을 조회하는 속성 액세스 연산자(`[]`)를 사용하여 이러한 제한을 쉽게 피할 수 있었습니다.

```
class PrivateExample
{
```

```
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);     // Strict 모드에서 컴파일 시 오류가 발생합니다 .
trace(myExample["privVar"]); // ActionScript 2.0 에서는 액세스가 허용되지만
  ActionScript 3.0 에서는 런타임 오류가 발생합니다 .
```

ActionScript 3.0의 경우 myExample.privVar과 같이 도트 연산자를 사용하여 전용 속성에 액세스하려고 하면 Strict 모드를 사용하는 경우 컴파일 타임 오류가 발생합니다. 그렇지 않은 경우에는 myExample["privVar"]과 같이 속성 액세스 연산자를 사용한 경우처럼 런타임에 오류가 보고됩니다.

다음 표에서는 동적이 아닌 봉인 클래스에 속한 전용 속성에 액세스한 결과를 보여 줍니다.

	Strict 모드	Standard 모드
도트 연산자(.)	컴파일 타임 오류	런타임 오류
대괄호 연산자([])	런타임 오류	런타임 오류

dynamic 특성으로 선언된 클래스의 경우에는 전용 변수에 액세스해도 런타임 오류가 발생하지 않습니다. 대신 변수를 참조할 수 없으므로 Flash Player에서 undefined 값이 반환됩니다. 그러나 Strict 모드에서 도트 연산자를 사용하면 컴파일 타임 오류가 발생합니다. 다음 예제는 이전 예제와 동일하지만 PrivateExample 클래스가 동적 클래스로 선언됩니다.

```
dynamic class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);     // Strict 모드에서 컴파일 시 오류가 발생합니다 .
trace(myExample["privVar"]); // 출력 : undefined
```

동적 클래스를 사용하면 클래스 외부 코드에서 전용 속성에 액세스할 때 일반적으로 오류가 발생하는 대신 undefined 값이 반환됩니다. 다음 표를 보면 Strict 모드에서 도트 연산자를 사용하여 전용 속성에 액세스할 때만 오류가 발생함을 알 수 있습니다.

	Strict 모드	Standard 모드
도트 연산자(.)	컴파일 타임 오류	undefined
대괄호 연산자([])	undefined	undefined

ActionScript 3.0에서 새로 도입된 protected 특성을 사용하면 자체 클래스나 하위 클래스에서 속성에 액세스할 수 있습니다. 즉, protected 속성은 자체 클래스 내에서 또는 상속 계층 구조에서 해당 클래스 아래쪽에 있는 모든 클래스에서 사용할 수 있습니다. 하위 클래스가 수퍼 클래스와 같은 패키지에 있는지 또는 다른 패키지에 있는지에 관계없이 하위 클래스에서 protected 속성을 사용할 수 있습니다.

ActionScript 2.0에 익숙한 사용자에게 이 기능은 ActionScript 2.0의 private 특성과 비슷합니다. ActionScript 3.0의 protected 특성은 Java의 protected 특성과도 비슷하지만 Java 버전의 경우 같은 패키지 내에 있는 호출자도 액세스할 수 있다는 차이점이 있습니다. protected 특성은 하위 클래스에 필요한 변수 또는 메서드를 상속 체인 외부에 있는 코드로부터 숨기려는 경우에 유용합니다.

ActionScript 3.0에서 새로 도입된 internal 특성을 사용하면 자체 패키지 내에서 속성을 호출할 수 있습니다. 이는 패키지 안에 있는 코드의 기본 특성이며 다음 특성이 지정되지 않은 모든 속성에 적용됩니다.

- public
- private
- protected
- 사용자 정의 네임스페이스

internal 특성은 Java의 기본 액세스 제어와 비슷하지만 Java의 경우에는 이 액세스 수준에 명시적으로 지정된 이름이 없으며 이 액세스 수준을 사용하려면 다른 액세스 수식어를 모두 생략해야 합니다. ActionScript 3.0에서 internal 특성을 사용하면 자체 패키지 내에서만 속성을 호출하도록 하려는 의도를 명시적으로 나타낼 수 있습니다.

static 특성

var, const 또는 function 키워드로 선언된 속성에 사용할 수 있는 static 특성을 통해 속성을 클래스의 인스턴스가 아닌 클래스 자체에 연결할 수 있습니다. 클래스 외부의 코드에서 정적 속성을 호출하려면 인스턴스 이름 대신 클래스 이름을 사용해야 합니다.

정적 속성은 하위 클래스로 상속되지는 않지만 하위 클래스의 범위 체인에 포함됩니다. 즉, 하위 클래스의 본문 내에서는 정적 변수 또는 메서드가 정의된 클래스를 참조하지 않고도 해당 변수 또는 메서드를 사용할 수 있습니다. 자세한 내용은 160페이지의 “상속되지 않는 정적 속성”을 참조하십시오.

사용자 정의 네임스페이스 특성

미리 정의된 액세스 제어 특성을 사용하는 대신 사용자 정의 네임스페이스를 만들어 특성으로 사용할 수 있습니다. 네임스페이스 특성은 정의마다 하나만 사용할 수 있으며 액세스 제어 특성(public, private, protected, internal)과는 함께 사용할 수 없습니다. 네임스페이스를 사용법에 대한 자세한 내용은 71페이지의 "네임스페이스"를 참조하십시오.

변수

변수는 var 또는 const 키워드로 선언할 수 있습니다. var 키워드로 선언된 변수의 값은 스크립트 실행 중 여러 차례 변경될 수 있습니다. const 키워드로 선언된 변수는 *상수*라고 하며, 값을 한 번만 지정할 수 있습니다. 초기화된 상수에 새 값을 지정하려고 하면 오류가 발생합니다. 자세한 내용은 102페이지의 "상수"를 참조하십시오.

정적 변수

정적 변수는 static 키워드를 var 또는 const 문 중 하나와 조합하여 선언합니다. 정적 변수는 클래스의 인스턴스가 아닌 클래스 자체에 연결되며, 객체의 전체 클래스에 적용되는 정보를 저장 및 공유하는 데 유용합니다. 예를 들어, 클래스가 인스턴스화된 횟수를 집계하거나 클래스 인스턴스의 최대 개수를 저장하려는 경우에 정적 변수가 적합합니다.

다음 예제에서는 totalCount 변수를 만들어 클래스 인스턴스화 횟수를 추적하고 MAX_NUM 상수를 만들어 인스턴스화 최대 횟수를 저장합니다. totalCount 및 MAX_NUM 변수는 특정 인스턴스가 아닌 전체 클래스에 적용되는 값을 포함하므로 정적 변수입니다.

```
class StaticVars
{
  public static var totalCount:int = 0;
  public static const MAX_NUM:uint = 16;
}
```

StaticVars 클래스의 외부에 있는 코드나 하위 클래스에 있는 코드에서는 클래스 자체를 통해서만 totalCount 및 MAX_NUM 속성을 참조할 수 있습니다. 예를 들어, 다음 코드를 사용해야 합니다.

```
trace(StaticVars.totalCount); // 출력 : 0
trace(StaticVars.MAX_NUM); // 출력 : 16
```

클래스 인스턴스를 통해서는 정적 변수에 액세스할 수 없으므로 다음 코드에서는 오류가 반환됩니다.

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // 오류
trace(myStaticVars.MAX_NUM); // 오류
```

static과 const 키워드를 모두 사용하여 선언한 변수는 StaticVars 클래스에서 MAX_NUM의 경우와 같이 상수를 선언함과 동시에 초기화해야 합니다. 생성자나 인스턴스 메서드 내에서는 MAX_NUM에 값을 지정할 수 없습니다. 다음 코드에서는 정적 상수를 제대로 초기화하지 않았으므로 오류가 발생합니다.

```
// !! 정적 상수를 이런 식으로 초기화하면 오류가 발생합니다 .
class StaticVars2
{
  public static const UNIQUESORT:uint;
  function initializeStatic():void
  {
    UNIQUESORT = 16;
  }
}
```

인스턴스 변수

인스턴스 변수에는 static 키워드 없이 var 및 const 키워드로 선언된 속성이 포함됩니다. 인스턴스 변수는 전체 클래스가 아닌 클래스 인스턴스에 연결되며 인스턴스에 고유한 값을 저장하는 데 유용합니다. 예를 들어 Array 클래스에는 Array 클래스의 특정 인스턴스에 들어 있는 배열 요소의 수가 저장되는 length라는 인스턴스 속성이 있습니다.

인스턴스 변수는 var 또는 const 등 선언된 방식에 관계없이 하위 클래스에서 재정의할 수 없습니다. 그러나 getter 및 setter 메서드를 재정의하면 변수를 재정의하는 것과 비슷한 결과를 달성할 수 있습니다. 자세한 내용은 145페이지의 “get 및 set 접근자 메서드”를 참조하십시오.

메서드

메서드는 클래스 정의에 포함되는 함수입니다. 클래스의 인스턴스를 만들면 해당 인스턴스에 메서드가 바인딩됩니다. 클래스 외부에 정의된 함수와 달리 메서드는 자신이 연결된 인스턴스와 분리하여 사용될 수 없습니다.

메서드는 function 키워드를 사용하여 정의됩니다. 다음과 같이 함수 명령문을 사용할 수 있습니다.

```
public function sampleFunction():String {}
```

다음과 같이 함수 표현식이 지정된 변수를 사용할 수도 있습니다.

```
public var sampleFunction:Function = function () {}
```

대부분의 경우에는 다음과 같은 이유로 인해 함수 표현식 대신 함수 명령문을 사용하는 것이 좋습니다.

- 함수 명령문은 보다 간결하고 쉽게 읽을 수 있습니다.
- 함수 명령문을 통해 override 및 final 키워드를 사용할 수 있습니다.
  자세한 내용은 158페이지의 “메서드 재정의”를 참조하십시오.

- 함수 명령문을 사용하면 식별자(함수 이름)와 메서드 본문 내의 코드가 보다 견고하게 결합됩니다. 대입문을 사용하면 변수 값을 변경할 수 있으므로 변수와 해당 함수 표현식 간의 연결은 언제든지 끊어질 수 있습니다. 변수를 var 대신 const로 선언하면 이 문제를 해결할 수는 있지만 코드를 읽기 어려워지고 override 및 final 키워드를 사용할 수 없게 되므로 이는 좋은 방법이 아닙니다.

프로토타입 객체에 함수를 연결하려는 경우에는 함수 표현식을 사용해야 합니다. 자세한 내용은 167페이지의 "프로토타입 객체"를 참조하십시오.

생성자 메서드

간단하게 *생성자*라고도 하는 생성자 메서드는 해당 메서드가 정의된 클래스와 이름이 같은 함수입니다. 생성자 메서드에 포함된 코드는 new 키워드를 사용하여 클래스의 인스턴스를 만들 때마다 실행됩니다. 예를 들어, 다음 코드에서는 status라는 속성 하나가 들어 있는 Example이라는 간단한 클래스를 정의합니다. status 변수의 초기값은 생성자 함수 안에서 설정됩니다.

```
class Example
{
  public var status:String;
  public function Example()
  {
    status = "initialized";
  }
}

var myExample:Example = new Example();
trace(myExample.status); // 출력 : initialized
```

생성자 메서드는 항상 공용이지만 public 특성은 생략할 수 있습니다. 생성자에는 private, protected 또는 internal 등의 다른 액세스 제어 지정자를 사용할 수 없습니다. 또한 생성자 메서드에는 사용자 정의 네임스페이스를 사용할 수 없습니다.

생성자에서는 super() 문을 사용하여 바로 위 수퍼 클래스의 생성자를 명시적으로 호출할 수 있습니다. 수퍼 클래스 생성자를 명시적으로 호출하지 않으면 컴파일러에서 생성자 본문의 첫 명령문 앞에 자동으로 호출을 삽입합니다. super 접두어로 수퍼 클래스를 참조하여 수퍼 클래스의 메서드를 호출할 수도 있습니다. 같은 생성자 본문에서 super()와 super를 함께 사용하려는 경우 super()를 먼저 호출해야 합니다. 그렇지 않으면 super 참조가 정상적으로 작동하지 않습니다. 또한 throw 또는 return 문을 사용하기 전에 super() 생성자를 호출해야 합니다.

다음 예제를 통해 super() 생성자를 호출하기 전에 super 참조를 사용한 결과를 확인해볼 수 있습니다. 새 클래스인 ExampleEx에서는 Example 클래스를 확장합니다. ExampleEx 생성자에서 super()를 호출하기 전에 수퍼 클래스에 정의된 status 변수에 액세스합니다. 이 경우 super() 생성자가 실행되기 전에는 status 변수를 사용할 수 없으므로 ExampleEx 생성자 내의 trace() 문에서 null 값을 출력합니다.

```
class ExampleEx extends Example
{
  public function ExampleEx()
  {
    trace(super.status);
    super();
  }
}

var mySample:ExampleEx = new ExampleEx(); // 출력 : null
```

생성자 내에서도 return 문을 사용할 수 있지만 값을 반환할 수는 없습니다. 즉, return 문에 표현식이나 값을 연결하면 안 됩니다. 따라서 생성자 메서드는 값을 반환할 수 없으며 반환 유형을 지정할 수 없습니다.

클래스에 생성자 메서드를 정의하지 않으면 컴파일러에서 자동으로 빈 생성자를 만듭니다. 클래스에서 다른 클래스를 확장하는 경우에는 컴파일러에서 자동으로 생성되는 생성자에 super() 호출이 포함됩니다.

정적 메서드

*클래스 메서드*라고도 하는 정적 메서드는 static 키워드로 선언되는 메서드입니다. 정적 메서드는 클래스 인스턴스가 아닌 클래스 자체에 연결되며, 개별 인스턴스가 아닌 전체적인 상태에 영향을 주는 기능을 캡슐화하는 데 유용합니다. 정적 메서드는 전체 클래스에 연결되므로 클래스의 인스턴스가 아닌 클래스를 통해서만 액세스할 수 있습니다.

정적 메서드는 클래스 인스턴스의 상태에 영향을 주는 데 국한되지 않는 기능을 캡슐화하는 데 유용합니다. 즉, 클래스 인스턴스의 값에 직접 영향을 주지 않는 기능을 제공하는 메서드는 정적 메서드로 선언해야 합니다. 예를 들어, Date 클래스에는 문자열을 받아 숫자로 변환하는 parse()라는 정적 메서드가 있습니다. 이 메서드는 클래스의 개별 인스턴스에 영향을 주지 않으므로 정적입니다. 대신 parse() 메서드는 날짜 값을 나타내는 문자열을 받아 파싱하여 Date 객체의 내부 표현과 호환되는 형식으로 숫자를 반환합니다. 이 메서드는 Date 클래스의 인스턴스에 직접 적용되지 않으므로 인스턴스 메서드가 아닙니다.

정적 parse() 메서드는 Date 클래스에 속하는 getMonth() 등의 인스턴스 메서드와 대조됩니다. getMonth() 메서드는 Date 인스턴스의 월을 나타내는 특정 구성 요소를 검색하여 인스턴스의 값에 직접 작용하므로 인스턴스 메서드입니다.

정적 메서드는 개별 인스턴스에 바인딩되지 않으므로 정적 메서드 본문 내에서는 this 또는 super 키워드를 사용할 수 없습니다. this 참조와 super 참조는 인스턴스 메서드의 컨텍스트 내에서만 의미가 있습니다.

몇 가지 다른 클래스 기반 프로그래밍 언어와 달리 ActionScript 3.0에서는 정적 메서드가 상속되지 않습니다. 자세한 내용은 160페이지의 "상속되지 않는 정적 속성"을 참조하십시오.

인스턴스 메서드

인스턴스 메서드는 static 키워드 없이 선언된 메서드입니다. 인스턴스 메서드는 전체 클래스가 아닌 클래스 인스턴스에 연결되며, 클래스의 개별 인스턴스에 영향을 주는 기능을 구현하는 데 유용합니다. 예를 들어 Array 클래스에는 Array 인스턴스에 직접 작용하는 sort()라는 인스턴스 메서드가 있습니다.

인스턴스 메서드 본문 내에서는 정적 변수와 인스턴스 변수가 모두 범위에 포함되므로 간단한 식별자를 사용하여 같은 클래스에 정의된 해당 변수를 참조할 수 있습니다. 예를 들어, 다음 CustomArray 클래스에서는 Array 클래스를 확장합니다. CustomArray 클래스에는 클래스 인스턴스의 총 개수를 추적하는 arrayCountTotal이라는 정적 변수, 인스턴스가 작성된 순서를 추적하는 arrayNumber라는 인스턴스 변수 및 이러한 변수의 값을 반환하는 getPosition()이라는 인스턴스 메서드가 정의됩니다.

```
public class CustomArray extends Array
{
  public static var arrayCountTotal:int = 0;
  public var arrayNumber:int;

  public function CustomArray()
  {
    arrayNumber = ++arrayCountTotal;
  }

  public function getArrayPosition():String
  {
     return ("Array " + arrayNumber + " of " + arrayCountTotal);
  }
}
```

클래스 외부의 코드에서는 CustomArray.arrayCountTotal과 같이 클래스 객체를 통해 정적 변수 arrayCountTotal을 참조해야 하지만, getPosition() 메서드 본문 내에 있는 코드에서는 정적 변수 arrayCountTotal을 직접 참조할 수 있습니다. 이는 수퍼 클래스에 있는 정적 변수의 경우에도 마찬가지입니다. ActionScript 3.0에서는 정적 속성이 상속되지 않지만 수퍼 클래스의 정적 속성은 범위에 포함됩니다. 예를 들어, Array 클래스에는 DESCENDING이라는 상수를 비롯한 몇 가지 정적 변수가 있습니다. Array의 하위 클래스에 있는 코드에서는 간단한 식별자를 사용하여 정적 상수 DESCENDING을 참조할 수 있습니다.

```
public class CustomArray extends Array
{
```

```
  public function testStatic():void
  {
    trace(DESCENDING); // 출력 : 2
  }
}
```

인스턴스 메서드의 본문 내에서 this 참조의 값은 해당 메서드가 연결된 인스턴스에 대한 참조입니다. 다음 코드에서는 this 참조가 해당 메서드가 포함된 인스턴스를 가리킴을 보여 줍니다.

```
class ThisTest
{
  function thisValue():ThisTest
  {
    return this;
  }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // 출력 : true
```

인스턴스 메서드의 상속은 override 및 final 키워드를 통해 제어할 수 있습니다. 상속된 메서드를 다시 정의하려면 override 특성을 사용하고, 하위 클래스에서 메서드를 재정의할 수 없게 하려면 final 특성을 사용합니다. 자세한 내용은 158페이지의 "메서드 재정의"를 참조하십시오.

get 및 set 접근자 메서드

getter 및 *setter*라고도 하는 get 및 set 접근자 함수를 통해 정보 은폐 및 캡슐화라는 프로그래밍 원칙을 준수하면서 클래스에 사용하기 쉬운 프로그래밍 인터페이스를 제공할 수 있습니다. get 및 set 함수를 사용하면 클래스 속성을 클래스 전용으로 유지하면서 클래스 사용자가 클래스 메서드를 호출하는 대신 클래스 변수에 액세스하는 것처럼 이러한 속성에 액세스하도록 허용할 수 있습니다.

이러한 방법을 사용하면 getPropertyName() 및 setPropertyName() 등 이름이 복잡한 기존 접근자 함수를 사용하지 않아도 된다는 장점이 있습니다. 또한 읽기 및 쓰기 액세스가 모두 허용되는 속성마다 두 개의 공용 함수를 만들지 않아도 됩니다.

GetSet이라는 다음 예제 클래스에는 privateProperty라는 전용 변수에 대한 액세스를 제공하는 publicAccess()라는 get 및 set 접근자 함수가 있습니다.

```
class GetSet
{
  private var privateProperty:String;

  public function get publicAccess():String
  {
    return privateProperty;
  }
```

```
  public function set publicAccess(setValue:String):void
  {
    privateProperty = setValue;
  }
}
```

다음과 같이 privateProperty 속성에 직접 액세스하려고 하면 오류가 발생합니다.

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // 오류가 발생합니다 .
```

GetSet 클래스 사용자는 속성처럼 보이는 publicAccess라는 항목을 대신 사용하게 되지만, 실제로 이는 privateProperty라는 전용 속성에 작용하는 get 및 set 접근자 함수 쌍입니다. 다음 예제에서는 GetSet 클래스를 인스턴스화한 다음 publicAccess라는 공용 접근자를 사용하여 privateProperty의 값을 설정합니다.

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // 출력 : null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // 출력 : hello
```

getter 및 setter 함수를 사용하여 수퍼 클래스에서 상속된 속성을 재정의할 수도 있습니다. 일반적인 클래스 멤버 변수를 사용할 때는 이러한 작업이 불가능합니다. var 키워드로 선언된 클래스 멤버 변수는 하위 클래스에서 재정의할 수 없습니다. 그러나 getter 및 setter 함수를 사용하여 만든 속성에는 이러한 제한이 없습니다. 수퍼 클래스에서 상속된 getter 및 setter 함수에 override 특성을 사용할 수 있습니다.

바인딩된 메서드

*메서드 클로저*라고도 하는 바인딩된 메서드는 해당 인스턴스에서 추출된 메서드입니다. 바인딩된 메서드의 예로는 함수에 인수로 전달된 메서드 또는 함수에서 값으로 반환된 메서드가 있습니다. ActionScript 3.0에 새로 도입된 바인딩된 메서드는 인스턴스에서 추출되어도 의미상의 환경이 유지된다는 점에서 함수 클로저와 비슷합니다. 그러나 바인딩된 메서드에서는 this 참조가 해당 메서드를 구현하는 인스턴스에 연결된 상태로 유지된다는 점에서 함수 클로저와 다릅니다. 즉, 바인딩된 메서드의 this 참조는 항상 메서드를 구현한 원래 객체를 가리킵니다. 함수 클로저의 경우에는 this 참조가 일반적이므로 호출 시점에서 함수에 연결된 객체를 가리킵니다.

this 키워드를 사용하는 경우 바인딩된 메서드를 잘 이해해야 합니다. this 키워드는 메서드의 부모 객체를 참조합니다. 대부분의 ActionScript 프로그래머는 this 키워드가 항상 메서드 정의가 들어 있는 객체 또는 클래스를 참조하리라고 예상합니다. 그러나 메서드 바인딩을 사용하지 않으면 그렇지 않을 수도 있습니다. 예를 들어, 이전 버전의 ActionScript에서는 this 참조가 메서드를 구현한 인스턴스를 참조하지 않을 수도 있었습니다. ActionScript 2.0의 경우 인스턴스에서 메서드를 추출하면 this 참조가 원래 인스턴스에 바인딩되지 않을 뿐만 아니라 인스턴스 클래스의 멤버 변수 및 메서드를 사용할 수 없습니다. ActionScript 3.0에서는 메서드를 매개 변수로 전달할 때 바인딩된 메서드가 자동으로 만들어지므로 이러한 문제가 없습니다. 바인딩된 메서드를 사용하면 this 키워드가 항상 메서드가 정의된 객체 또는 클래스를 참조하게 됩니다.

다음 코드에서는 ThisTest라는 클래스를 정의합니다. 이 클래스에는 바인딩된 메서드를 정의하는 foo()라는 메서드 및 바인딩된 메서드를 반환하는 bar()라는 메서드가 있습니다. 클래스 외부에 있는 코드에서는 ThisTest 클래스의 인스턴스를 만들고 bar() 메서드를 호출한 다음 반환 값을 myFunc라는 변수에 저장합니다.

```
class ThisTest
{
  private var num:Number = 3;
  function foo():void // 바인딩된 메서드 정의됨
  {
    trace("foo's this: " + this);
    trace("num: " + num);
  }
  function bar():Function
  {
    return foo; // 바인딩된 메서드 반환됨
  }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // 출력 : [object global]
myFunc();
/* 출력 :
foo's this: [object ThisTest]
출력 : num: 3 */
```

코드의 마지막 두 행을 보면 this 참조는 전역 객체를 가리키지만 바인딩된 메서드인 foo()의 this 참조는 계속 ThisTest 클래스를 가리키는 것을 알 수 있습니다. 또한 myFunc 변수에 저장된 바인딩된 메서드에서는 ThisTest 클래스의 멤버 변수에 계속 액세스할 수 있습니다. 이 코드를 ActionScript 2.0에서 실행하면 this 참조가 서로 일치하게 되고 num 변수는 undefined가 됩니다.

바인딩된 메서드를 사용하는 것이 특히 유용한 부분은 이벤트 핸들러입니다. addEventListener() 메서드에는 함수 또는 메서드를 인수로 전달해야 하기 때문입니다. 자세한 내용은 310페이지의 “클래스 메서드로 정의되는 리스너 함수”를 참조하십시오.

클래스와 열거형

*열거형*은 소규모 값 집합을 캡슐화하기 위해 만드는 사용자 정의 데이터 유형입니다. ActionScript 3.0에서는 C++의 `enum` 키워드나 Java의 Enumeration 인터페이스와 같은 구체적인 열거형 기능이 지원되지 않습니다. 그러나 클래스와 정적 상수를 사용하여 열거형을 만들 수 있습니다. 예를 들어 Flash Player API의 PrintJob 클래스에서는 PrintJobOrientation이라는 열거형을 사용하여 다음 코드와 같이 `"landscape"`와 `"portrait"`으로 구성된 값 집합을 저장합니다.

```
public final class PrintJobOrientation
{
  public static const LANDSCAPE:String = "landscape";
  public static const PORTRAIT:String = "portrait";
}
```

열거형 클래스는 확장할 필요가 없으므로 관습적으로 `final` 특성으로 선언됩니다. 이러한 클래스는 정적 멤버만으로 구성되므로 클래스의 인스턴스를 만들 수 없습니다. 대신 다음 인용 코드와 같이 클래스 객체를 직접 사용하여 열거형 값에 액세스합니다.

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
  if (pj.orientation == PrintJobOrientation.PORTRAIT)
  {
    ...
  }
  ...
}
```

Flash Player API의 모든 열거형 클래스에는 String, int 또는 uint 유형의 변수만 들어 있습니다. 리터럴 문자열이나 숫자 값 대신 열거형을 사용하면 철자를 잘못 입력한 오류를 쉽게 찾을 수 있다는 장점이 있습니다. 열거형의 이름을 잘못 입력하면 ActionScript 컴파일러에서 오류가 발생합니다. 리터럴 값을 사용하면 단어의 철자를 잘못 입력하거나 숫자를 잘못 입력해도 컴파일러에서 메시지가 표시되지 않습니다. 이전 예제의 경우 다음 인용 코드와 같이 열거형 상수 이름을 잘못 입력하면 컴파일러에서 오류가 발생합니다.

```
  if (pj.orientation == PrintJobOrientation.PORTRAI) // 컴파일러 오류
```

그러나 다음과 같이 문자열 리터럴 값을 잘못 입력한 경우에는 컴파일러에서 오류가 발생하지 않습니다.

```
  if (pj.orientation == "portrai") // 컴파일러 오류 없음
```

열거형을 만드는 두 번째 방법에서도 열거형에 사용할 정적 속성이 있는 별도의 클래스를 만듭니다. 그러나 두 번째 방법은 각 정적 속성에 문자열 또는 정수 값 대신 클래스 인스턴스가 포함된다는 점이 다릅니다. 예를 들어, 다음 코드에서는 요일에 대한 열거형 클래스를 만듭니다.

```
public final class Day
{
  public static const MONDAY:Day = new Day();
  public static const TUESDAY:Day = new Day();
  public static const WEDNESDAY:Day = new Day();
  public static const THURSDAY:Day = new Day();
  public static const FRIDAY:Day = new Day();
  public static const SATURDAY:Day = new Day();
  public static const SUNDAY:Day = new Day();
}
```

Flash Player API에는 이 기술이 사용되지 않지만, 이 기술을 통해 제공되는 향상된 유형 확인을 선호하는 많은 개발자는 이 기술을 사용하고 있습니다. 예를 들어, 열거형 값을 반환하는 메서드에서는 반환 값을 열거형 데이터 유형으로 제한할 수 있습니다. 다음 코드에서는 요일을 반환하는 함수뿐만 아니라 열거형을 유형 약어로 사용하는 함수 호출도 보여 줍니다.

```
function getDay():Day
{
  var date:Date = new Date();
  var retDay:Day;
  switch (date.day)
  {
    case 0:
      retDay = Day.MONDAY;
      break;
    case 1:
      retDay = Day.TUESDAY;
      break;
    case 2:
      retDay = Day.WEDNESDAY;
      break;
    case 3:
      retDay = Day.THURSDAY;
      break;
    case 4:
      retDay = Day.FRIDAY;
      break;
    case 5:
      retDay = Day.SATURDAY;
      break;
    case 6:
      retDay = Day.SUNDAY;
      break;
  }
  return retDay;
}

var dayOfWeek:Day = getDay();
```

Day 클래스를 개선하여 각 요일을 정수에 연결하고 문자열로 나타낸 요일을 반환하는 `toString()` 메서드를 제공할 수도 있습니다. 연습을 위해 Day 클래스를 이러한 방식으로 개선해 볼 수 있습니다.

포함된 에셋 클래스

ActionScript 3.0에서는 *포함된 에셋 클래스*라는 특수 클래스를 사용하여 포함된 에셋을 나타냅니다. *포함된 에셋*은 컴파일 타임에 SWF 파일에 포함된 사운드, 이미지 또는 글꼴 등의 에셋입니다. 에셋을 동적으로 로드하지 않고 에셋을 포함하면 런타임에 항상 에셋을 사용할 수 있다는 장점이 있지만 SWF 파일 크기가 증가하는 단점이 있습니다.

Flash에서 포함된 에셋 클래스 사용

에셋을 포함하려면 먼저 FLA 파일의 라이브러리에 에셋을 배치합니다. 그런 다음 에셋의 링크 속성을 사용하여 포함된 에셋 클래스에 이름을 지정합니다. 이 이름을 가진 클래스가 클래스 경로에 없는 경우 사용자가 지정한 이름의 에셋이 자동으로 생성됩니다. 그런 다음 포함된 에셋 클래스의 인스턴스를 만든 후, 이 클래스에 의해 정의되거나 상속된 속성 및 메서드를 사용합니다. 예를 들어 다음과 같은 코드를 사용하면 포함된 에셋 클래스 PianoMusic에 연결된 포함된 사운드를 재생할 수 있습니다.

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

인터페이스

인터페이스는 관련되지 않은 여러 객체가 서로 통신할 수 있게 하는 메서드 선언을 모아 놓은 것입니다. 예를 들어, Flash Player API에는 클래스에서 이벤트 객체를 처리하는 데 사용할 수 있는 메서드 선언이 들어 있는 IEventDispatcher 인터페이스가 정의되어 있습니다. 여러 객체는 IEventDispatcher 인터페이스를 통해 표준화된 방식으로 이벤트 객체를 서로 전달할 수 있습니다. 다음 코드에서는 IEventDispatcher 인터페이스의 정의를 보여 줍니다.

```
public interface IEventDispatcher
{
  function addEventListener(type:String, listener:Function,
       useCapture:Boolean=false, priority:int=0,
       useWeakReference:Boolean = false):void;
  function removeEventListener(type:String, listener:Function,
       useCapture:Boolean=false):void;
  function dispatchEvent(event:Event):Boolean;
  function hasEventListener(type:String):Boolean;
  function willTrigger(type:String):Boolean;
}
```

인터페이스는 메서드의 인터페이스와 해당 구현이 서로 구분된다는 점을 기반으로 합니다. 메서드의 인터페이스에는 해당 메서드를 호출하는 데 필요한 메서드 이름, 모든 매개 변수 및 반환 유형 등의 모든 정보가 포함됩니다. 메서드 구현에는 인터페이스 정보뿐만 아니라 메서드의 비헤이비어를 수행하는 실행 가능한 명령문도 포함됩니다. 인터페이스 정의에는 메서드 인터페이스만 들어 있으며, 해당 인터페이스를 구현하는 모든 클래스에서는 메서드 구현을 정의해야 합니다.

Flash Player API의 EventDispatcher 클래스에서는 모든 IEventDispatcher 인터페이스 메서드를 정의하고 각 메서드에 본문을 추가하여 IEventDispatcher 인터페이스를 구현합니다. 다음은 EventDispatcher 클래스 정의에서 인용한 코드입니다.

```
public class EventDispatcher implements IEventDispatcher
{
  function dispatchEvent(event:Event):Boolean
  {
    /* 구현 명령문 */
  }

  ...
}
```

IEventDispatcher 인터페이스는 EventDispatcher 인스턴스에서 이벤트 객체를 처리하고 마찬가지로 IEventDispatcher 인터페이스를 구현하는 다른 객체로 이벤트 객체를 전달하는 데 사용되는 프로토콜 역할을 합니다.

또한 인터페이스는 클래스와 마찬가지로 데이터 유형을 정의합니다. 따라서 클래스와 마찬가지로 인터페이스를 유형 약어로 사용할 수 있습니다. 인터페이스는 데이터 유형이므로 데이터 유형이 필요한 `is` 및 `as` 등의 연산자와 함께 사용할 수 있습니다. 그러나 클래스와 달리 인터페이스는 인스턴스화할 수 없습니다. 이러한 차이점으로 인해 인터페이스를 추상적인 데이터 유형으로, 클래스를 구체적인 데이터 유형으로 간주하는 프로그래머가 많습니다.

인터페이스 정의

인터페이스 정의의 구조는 클래스 정의의 구조와 비슷하지만 인터페이스에는 본문이 없는 메서드만 포함될 수 있습니다. 인터페이스에는 변수 또는 상수가 포함될 수 없지만 getter 및 setter는 포함될 수 있습니다. 인터페이스를 정의하려면 `interface` 키워드를 사용합니다. 예를 들어, 다음 IExternalizable 인터페이스는 Flash Player API의 flash.utils 패키지에 속합니다. IExternalizable 인터페이스에서는 객체를 직렬화하는 프로토콜을 정의합니다. 직렬화란 객체를 장치에 저장하거나 네트워크를 통해 전송하는 데 적합한 형식으로 변환하는 것입니다.

```
public interface IExternalizable
{
  function writeExternal(output:IDataOutput):void;
  function readExternal(input:IDataInput):void;
}
```

IExternalizable 인터페이스는 public 액세스 제어 수정자와 함께 선언되어 있습니다. 인터페이스 정의에는 public 및 internal 액세스 제어 지정자만 사용할 수 있습니다. 인터페이스 정의 내부의 메서드 선언에는 액세스 제어 지정자를 사용할 수 없습니다.

Flash Player API에서는 인터페이스 이름을 대문자 I로 시작하는 규칙을 따르고 있지만, 모든 유효한 식별자를 인터페이스 이름으로 사용할 수 있습니다. 인터페이스 정의는 패키지의 최상위 수준에 배치되는 경우가 많습니다. 클래스 정의 내부나 다른 인터페이스 정의 내부에는 인터페이스 정의를 배치할 수 없습니다.

인터페이스는 하나 이상의 다른 인터페이스를 확장할 수 있습니다. 예를 들어 다음 IExample 인터페이스에서는 IExternalizable 인터페이스를 확장합니다.

```
public interface IExample extends IExternalizable
{
  function extra():void;
}
```

IExample 인터페이스를 구현하는 모든 클래스에는 extra() 메서드뿐만 아니라 IExternalizable 인터페이스에서 상속된 writeExternal() 및 readExternal() 메서드에 대한 구현도 포함되어야 합니다.

클래스에서 인터페이스 구현

클래스는 인터페이스를 구현할 수 있는 유일한 ActionScript 3.0 언어 요소입니다. 하나 이상의 인터페이스를 구현하려면 클래스 선언에 implements 키워드를 사용합니다. 다음 예제에서는 IAlpha 및 IBeta라는 두 인터페이스를 정의하고, 두 인터페이스를 모두 구현하는 Alpha라는 클래스를 정의합니다.

```
interface IAlpha
{
  function foo(str:String):String;
}

interface IBeta
{
  function bar():void;
}

class Alpha implements IAlpha, IBeta
{
  public function foo(param:String):String {}
  public function bar():void {}
}
```

인터페이스를 구현하는 클래스에서 구현된 메서드는 다음 조건에 맞아야 합니다.

- `public` 액세스 제어 식별자를 사용해야 합니다.
- 인터페이스 메서드와 같은 이름을 사용해야 합니다.
- 매개 변수 개수가 같아야 하고 각 매개 변수의 데이터 유형이 인터페이스 메서드 매개 변수의 데이터 유형과 일치해야 합니다.
- 같은 반환 유형을 사용해야 합니다.

그러나 구현하는 메서드의 매개 변수 이름은 융통성 있게 지정할 수 있습니다. 구현된 메서드에서 매개 변수 개수 및 각 매개 변수의 데이터 유형은 인터페이스 메서드와 일치해야 하지만 매개 변수 이름은 일치하지 않아도 됩니다. 예를 들어, 이전 예제에서 `Alpha.foo()` 메서드의 매개 변수 이름은 `param`입니다.

```
public function foo(param:String):String {}
```

그러나 `IAlpha.foo()` 인터페이스 메서드에서는 매개 변수 이름이 `str`입니다.

```
function foo(str:String):String;
```

매개 변수 기본값도 유연하게 지정할 수 있습니다. 인터페이스 정의에는 매개 변수 기본값이 지정된 함수 선언이 포함될 수 있습니다. 이러한 함수 선언을 구현하는 메서드에는 인터페이스 정의에 지정된 값과 데이터 유형이 같은 매개 변수 기본값이 있어야 하지만 실제 값은 일치하지 않아도 됩니다. 예를 들어, 다음 코드에서는 매개 변수 기본값이 3인 메서드가 들어 있는 인터페이스를 정의합니다.

```
interface IGamma
{
  function doSomething(param:int = 3):void;
}
```

다음 클래스 정의에서는 Igamma 인터페이스를 구현하지만 다른 매개 변수 기본값을 사용합니다.

```
class Gamma implements IGamma
{
  public function doSomething(param:int = 4):void {}
}
```

이러한 유연성을 발휘할 수 있는 이유는, 인터페이스 구현에 대한 규칙이 데이터 유형의 호환성을 보장하기 위해 설계되었으며 이를 위해 매개 변수 이름과 매개 변수 기본값이 항상 일치할 필요는 없기 때문입니다.

상속

상속은 코드 재사용의 한 형태로서, 프로그래머는 이를 통해 기존 클래스를 기반으로 새 클래스를 개발할 수 있습니다. 기존 클래스는 일반적으로 *기본 클래스* 또는 *수퍼 클래스*라고 하며 새 클래스는 일반적으로 *하위 클래스*라고 합니다. 상속의 가장 중요한 장점은 기존 코드를 그대로 두면서 기본 클래스의 코드를 재사용할 수 있다는 점입니다. 또한 상속을 사용해도 다른 클래스와 기본 클래스가 상호 작용하는 방식은 변경할 필요가 없습니다. 완벽하게 테스트되었거나 이미 사용 중인 기존 클래스를 수정하는 대신 상속을 사용하면 해당 클래스를 통합된 모듈로 취급하면서 속성 또는 메서드를 추가하여 확장할 수 있습니다. 따라서 클래스가 다른 클래스에서 상속됨을 나타내려면 `extends` 키워드를 사용합니다.

또한 상속을 통해 코드에서 *다형성*의 장점을 활용할 수 있습니다. 다형성이란 다양한 데이터 유형에 이름이 같은 메서드를 적용하여 서로 다른 결과를 얻을 수 있는 기능입니다. 간단한 예제로, Shape라는 기본 클래스와 Circle 및 Square라는 하위 클래스 두 개를 가정해 봅니다. Shape 클래스에는 도형의 넓이를 반환하는 `area()`라는 메서드가 정의됩니다. 다형성을 구현하면 Circle 유형 객체와 Square 유형 객체를 대상으로 `area()` 메서드를 호출하여 자동으로 올바른 계산 결과를 얻을 수 있습니다. 상속을 사용하면 하위 클래스에서 기본 클래스의 메서드를 상속하여 다시 정의(*재정의*)할 수 있으므로 다형성을 구현할 수 있습니다. 다음 예제에서는 Circle 및 Square 클래스에 의해 `area()` 메서드가 다시 정의됩니다.

```
class Shape
{
  public function area():Number
  {
    return NaN;
  }
}

class Circle extends Shape
{
  private var radius:Number = 1;
  override public function area():Number
  {
    return (Math.PI * (radius * radius));
  }
}

class Square extends Shape
{
  private var side:Number = 1;
  override public function area():Number
  {
    return (side * side);
  }
}
```

```
var cir:Circle = new Circle();
trace(cir.area()); // 출력 : 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // 출력 : 1
```

각 클래스는 데이터 유형을 정의하므로 상속을 사용하면 기본 클래스와 이를 확장하는 클래스 사이에 특수한 관계가 성립됩니다. 하위 클래스는 항상 기본 클래스의 모든 속성을 가지게 되므로 하위 클래스의 인스턴스를 항상 기본 클래스의 인스턴스 대신 사용할 수 있습니다. 예를 들어, 메서드에 Shape 유형 매개 변수가 정의된 경우 Circle은 Shape를 확장하므로 다음과 같이 Circle 유형 인수를 전달할 수 있습니다.

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

인스턴스 속성과 상속

function, var 또는 const 등 정의에 사용된 키워드에 관계없이 인스턴스 속성은 기본 클래스에서 private 특성으로 선언되어 있지 않으면 모든 하위 클래스에 상속됩니다. 예를 들어, Flash Player API의 Event 클래스에는 모든 이벤트 객체에 공통되는 속성을 상속하는 여러 하위 클래스가 있습니다.

일부 이벤트 유형의 경우에는 Event 클래스에 이벤트를 정의하는 데 필요한 모든 속성이 들어 있습니다. 이러한 이벤트 유형에는 Event 클래스에 정의된 인스턴스 속성 이외에 추가 속성이 필요하지 않습니다. 이러한 이벤트의 예로는 데이터가 성공적으로 로드될 때 발생하는 complete 이벤트 및 네트워크 연결이 설정될 때 발생하는 connect 이벤트가 있습니다.

다음은 Event 클래스에서 인용한 코드로서 하위 클래스로 상속되는 속성 및 메서드 중 일부를 보여 줍니다. 이러한 속성은 상속되므로 모든 하위 클래스의 인스턴스에서 속성에 액세스할 수 있습니다.

```
public class Event
{
  public function get type():String;
  public function get bubbles():Boolean;
  ...

  public function stopPropagation():void {}
  public function stopImmediatePropagation():void {}
  public function preventDefault():void {}
  public function isDefaultPrevented():Boolean {}
  ...
}
```

Event 클래스에 없는 고유 속성이 필요한 이벤트 유형도 있습니다. 이러한 이벤트는 Event 클래스에 정의된 속성에 새 속성을 추가할 수 있도록 Event 클래스의 하위 클래스를 통해 정의됩니다. 이러한 하위 클래스의 예로는 마우스 이동이나 마우스 클릭에 관련된 mouseMove 및 click 등의 이벤트에 고유한 속성을 추가하는 MouseEvent 클래스가 있습니다. 다음은 MouseEvent 클래스에서 인용한 코드로서 기본 클래스에는 없고 하위 클래스에만 있는 속성의 정의를 보여 줍니다.

```
public class MouseEvent extends Event
{
  public static const CLICK:String      = "click";
  public static const MOUSE_MOVE:String = "mouseMove";
  ...

  public function get stageX():Number {}
  public function get stageY():Number {}
  ...
}
```

액세스 제어 지정자와 상속

public 키워드로 선언된 속성은 코드의 모든 위치에서 참조할 수 있습니다. 즉, public 키워드는 private, protected 및 internal 키워드와 달리 속성 상속에 제한이 없습니다.

private 키워드로 선언된 속성은 해당 속성이 정의된 클래스 내에서만 참조할 수 있으므로 하위 클래스로 상속되지 않습니다. 이 비헤이비어는 private 키워드가 ActionScript 3.0의 protected 키워드와 비슷하게 작동했던 이전 버전의 ActionScript와 다릅니다.

protected 키워드는 속성이 정의된 클래스뿐만 아니라 모든 하위 클래스에서도 속성을 참조할 수 있음을 나타냅니다. Java 프로그래밍 언어의 protected 키워드와 달리 ActionScript 3.0에서는 protected 키워드를 사용하는 경우 같은 패키지의 다른 모든 클래스에서 해당 속성을 참조할 수 없습니다. ActionScript 3.0에서는 하위 클래스에서만 protected 키워드로 선언된 속성에 액세스할 수 있습니다. 또한 하위 클래스가 기본 클래스와 같은 패키지에 있는지 또는 다른 패키지에 있는지에 관계없이 하위 클래스에서 protected 속성을 참조할 수 있습니다.

속성이 정의된 패키지 내에서만 속성을 참조할 수 있게 하려면 internal 키워드를 사용하거나 액세스 제어 지정자를 사용하지 않습니다. internal 액세스 제어 지정자는 키워드가 지정되지 않은 경우에 적용되는 기본 액세스 제어 지정자입니다. internal로 표시된 속성은 같은 패키지 안에 있는 하위 클래스에만 상속됩니다.

다음 예제를 통해 각 액세스 제어 지정자가 패키지 경계를 벗어나는 상속에 주는 영향을 확인할 수 있습니다. 다음 코드에서는 AccessControl이라는 기본 응용 프로그램 클래스 및 Base와 Extender라는 기타 클래스 두 개를 정의합니다. Base 클래스는 foo라는 패키지에 있고, Base 클래스의 하위 클래스인 Extender 클래스는 bar라는 패키지에 있습니다. AccessControl 클래스에서는 Extender 클래스만 가져와서 Extender의 인스턴스를 만들고, 이 인스턴스에서는 Base 클래스에 정의된 str이라는 변수에 액세스합니다. str 변수는 public으로 선언되어 있으므로 이 코드는 다음과 같이 컴파일 및 실행됩니다.

```
// foo라는 폴더의 Base.as
package foo
{
  public class Base
  {
    public var str:String = "hello"; // 이 행에서 public을 변경합니다.
  }
}

// bar라는 폴더의 Extender.as
package bar
{
  import foo.Base;
  public class Extender extends Base
  {
    public function getString():String {
      return str;
    }
  }
}

// ProtectedExample.as라는 파일의 기본 응용 프로그램 클래스
  import flash.display.MovieClip;
  import bar.Extender;
  public class AccessControl extends MovieClip
  {
    public function AccessControl()
    {
      var myExt:Extender = new Extender();
      trace(myExt.testString);  // str이 public이 아닌 경우 오류가 발생합니다.
      trace(myExt.getString()); // str이 private 또는 internal인 경우 오류가
  발생합니다.
    }
  }
}
```

다른 액세스 제어 지정자가 이전 예제의 컴파일 및 실행에 주는 영향을 확인하려면 AccessControl 클래스에서 다음 줄을 삭제하거나 주석 처리한 후 str 변수의 액세스 제어 지정자를 private, protected 또는 internal로 변경해 봅니다.

```
trace(myExt.testString);  // str이 public이 아닌 경우 오류
```

변수는 재정의할 수 없음

var 또는 const 키워드로 선언된 속성은 상속되지만 재정의할 수는 없습니다. 속성을 재정의한다는 것은 하위 클래스에서 속성을 다시 정의함을 의미합니다. 재정의할 수 있는 속성 유형은 function 키워드로 선언된 속성인 메서드뿐입니다. 인스턴스 변수는 재정의할 수 없지만 인스턴스 변수에 대한 getter 및 setter 메서드를 만들고 해당 메서드를 재정의하면 비슷한 기능을 달성할 수 있습니다. 자세한 내용은 159페이지의 "getter 및 setter 재정의"를 참조하십시오.

메서드 재정의

메서드를 재정의한다는 것은 상속된 메서드의 비헤이비어를 다시 정의함을 의미합니다. 정적 메서드는 상속되지 않으며 재정의할 수 없습니다. 그러나 인스턴스 메서드는 하위 클래스로 상속되며 다음 두 가지 조건에 맞는 경우 재정의할 수 있습니다.

- 인스턴스 메서드가 기본 클래스에서 final 키워드로 선언되어 있지 않아야 합니다. 인스턴스 메서드에 final 키워드가 사용되면 하위 클래스에서 메서드를 재정의할 수 없습니다.
- 인스턴스 메서드가 기본 클래스에서 private 액세스 제어 지정자로 선언되어 있지 않아야 합니다. 메서드가 기본 클래스에서 private으로 표시된 경우에는 하위 클래스에서 기본 클래스 메서드를 참조할 수 없으므로 이름이 같은 메서드를 정의할 때 override 키워드를 사용할 필요가 없습니다.

이러한 조건에 맞는 인스턴스 메서드를 재정의하려면 하위 클래스의 메서드 정의에서 override 키워드를 사용해야 하고 메서드의 수퍼 클래스 버전과 다음과 같은 점에서 일치해야 합니다.

- 재정의 메서드의 액세스 제어 수준이 기본 클래스 메서드와 일치해야 합니다. internal로 표시된 메서드의 액세스 제어 수준은 액세스 제어 지정자가 없는 메서드와 같습니다.
- 재정의 메서드의 매개 변수 개수가 기본 클래스 메서드와 일치해야 합니다.
- 재정의 메서드 매개 변수의 데이터 유형 약어가 기본 클래스 메서드의 매개 변수와 일치해야 합니다.
- 재정의 메서드의 반환 유형이 기본 클래스 메서드와 일치해야 합니다.

그러나 재정의 메서드의 매개 변수 이름은 기본 클래스의 매개 변수 이름과 일치하지 않아도 되며, 매개 변수 개수와 각 매개 변수의 데이터 유형만 일치하면 됩니다.

super 문

프로그래머는 메서드를 재정의할 때 기존 비헤이비어를 완전히 대체하기보다는 해당 수퍼 클래스 메서드의 비헤이비어에 기능을 추가하려는 경우가 많습니다. 이렇게 하려면 하위 클래스의 메서드에서 자신의 수퍼 클래스 버전을 호출할 수 있는 메커니즘이 필요합니다. `super` 문은 바로 위 수퍼 클래스를 참조하므로 이러한 메커니즘을 지원합니다. 다음 예제에서는 `thanks()`라는 메서드가 포함된 Base라는 클래스를 정의하고, Base 클래스의 하위 클래스이며 `thanks()` 메서드를 재정의하는 Extender 클래스를 정의합니다. `Extender.thanks()` 메서드에서는 `super` 문을 사용하여 `Base.thanks()`를 호출합니다.

```
package {
  import flash.display.MovieClip;
  public class SuperExample extends MovieClip
  {
    public function SuperExample()
    {
      var myExt:Extender = new Extender()
      trace(myExt.thanks()); // 출력 : Mahalo nui loa
    }
  }
}

class Base {
  public function thanks():String
  {
    return "Mahalo";
  }
}

class Extender extends Base
{
  override public function thanks():String
  {
    return super.thanks() + " nui loa";
  }
}
```

getter 및 setter 재정의

수퍼 클래스에 정의된 변수는 재정의할 수 없지만 getter 및 setter는 재정의할 수 있습니다. 예를 들어 다음 코드에서는 Flash Player API의 MovieClip 클래스에 정의되어 있는 `currentLabel`이라는 getter를 재정의합니다.

```
package
{
  import flash.display.MovieClip;
  public class OverrideExample extends MovieClip
  {
    public function OverrideExample()
```

```
    {
      trace(currentLabel)
    }
    override public function get currentLabel():String
    {
      var str:String = "Override: ";
      str += super.currentLabel;
      return str;
    }
  }
}
```

OverrideExample 클래스 생성자에 있는 `trace()` 문에서는 `Override: null`이 산출되며, 이는 `currentLabel` 속성이 예제에서 재정의되었음을 보여 줍니다.

상속되지 않는 정적 속성

정적 속성은 하위 클래스로 상속되지 않습니다. 즉, 하위 클래스의 인스턴스를 통해서는 정적 속성에 액세스할 수 없습니다. 정적 속성은 해당 속성이 정의된 클래스 객체를 통해서만 액세스할 수 있습니다. 예를 들어, 다음 코드에서는 Base라는 기본 클래스와 Base를 확장하는 Extender라는 하위 클래스를 정의합니다. Base 클래스에는 `test`라는 정적 변수가 정의됩니다. 다음 인용 코드는 Strict 모드의 경우 컴파일되지 않으며 Standard 모드의 경우 런타임 오류가 발생합니다.

```
package {
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // 오류
    }
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base { }
```

정적 변수 `test`에 액세스하려면 다음 코드와 같이 클래스 객체를 사용해야 합니다.

```
Base.test;
```

그러나 정적 속성과 이름이 같은 인스턴스 속성을 정의할 수는 있습니다. 이러한 인스턴스 속성은 동일한 클래스에 정적 속성과 함께 정의하거나 하위 클래스에 정의할 수 있습니다. 예를 들어, 이전 예제의 Base 클래스에 test라는 인스턴스 속성이 있을 수 있습니다. 다음 코드에서는 인스턴스 속성이 Extender 클래스로 상속되므로 코드가 정상적으로 컴파일 및 실행됩니다. test 인스턴스 변수의 정의를 잘라내어 Extender 클래스에 붙여 넣어도 코드가 정상적으로 컴파일 및 실행됩니다.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // 출력 : instance
    }
  }
}

class Base
{
  public static var test:String = "static";
  public var test:String = "instance";
}

class Extender extends Base {}
```

정적 속성과 범위 체인

정적 속성은 상속되지 않지만, 해당 속성이 정의된 클래스 및 해당 클래스에 대한 모든 하위 클래스의 범위 체인에 포함됩니다. 따라서 정적 속성은 자신이 정의된 클래스 및 모든 하위 클래스의 *범위 내*에 있습니다. 즉, 정적 속성이 정의된 클래스 본문 및 해당 클래스의 모든 하위 클래스 내에서 정적 속성에 직접 액세스할 수 있습니다.

다음 예제에서는 이전 예제에서 정의한 클래스를 수정하여 Base 클래스에 정의된 정적 변수 test가 Extender 클래스의 범위 내에 있음을 보여 줍니다. 즉, Extender 클래스에서는 test가 정의된 클래스의 이름을 접두어로 사용하지 않고도 정적 변수 test에 액세스할 수 있습니다.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
```

```
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base
{
  public function Extender()
  {
    trace(test); // 출력 : static
  }

}
```

같은 클래스나 하위 클래스에 정적 속성과 이름이 같은 인스턴스 속성이 정의된 경우 범위 체인에서 인스턴스 속성의 우선 순위가 더 높습니다. 따라서 인스턴스 속성이 정적 속성을 *가리게* 되어 정적 속성 값 대신 인스턴스 속성 값이 사용됩니다. 예를 들어 다음 코드에서는 Extender 클래스에 `test`라는 인스턴스 변수가 정의되면 `trace()` 문에 정적 변수 값 대신 인스턴스 변수 값이 사용됨을 보여 줍니다.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base
{
  public static var test:String = "static";
}

class Extender extends Base
{
  public var test:String = "instance";
  public function Extender()
  {
    trace(test); // 출력 : instance
  }

}
```

고급 항목

이 단원에서는 우선 ActionScript와 OOP의 역사를 간략하게 소개한 다음, ActionScript 3.0 객체 모델에 대해 설명하고 이를 통해 새로운 AVM2(ActionScript Virtual Machine 2)를 도입하여 AVM1(ActionScript Virtual Machine 1)이 포함된 이전 버전의 Flash Player보다 크게 향상된 성능을 갖게 됨을 설명합니다.

ActionScript OOP 지원 이력

ActionScript 3.0은 이전 버전의 ActionScript를 기초로 제작되었으므로 ActionScript 객체 모델의 발전 과정을 이해하면 도움이 됩니다. ActionScript는 초기 버전의 Flash 제작 도구를 위한 간단한 스크립트 메커니즘으로 처음 개발되었습니다. 이후 프로그래머들은 ActionScript로 점차 복잡한 응용 프로그램을 작성하기 시작했습니다. 이러한 프로그래머의 수요에 대응하기 위해 이후 릴리스에서는 복잡한 응용 프로그램을 만드는 데 필요한 언어 기능이 계속 추가되었습니다.

ActionScript 1.0

ActionScript 1.0은 Flash Player 6 및 이전 버전에서 사용된 언어 버전입니다. 이러한 초기 개발 단계에서도 ActionScript 객체 모델은 기초적인 데이터 유형으로서 객체라는 개념에 기반을 두었습니다. ActionScript 객체는 *속성* 그룹이 있는 복합 데이터 유형입니다. 객체 모델에서 *속성*이라는 용어에는 객체에 연결된 변수, 함수 또는 메서드 등의 모든 항목이 포함됩니다.

이러한 1세대 ActionScript에서는 `class` 키워드를 통한 클래스 정의가 지원되지 않지만 프로토타입 객체라는 특수한 객체를 사용하여 클래스를 정의할 수 있습니다. Java 및 C++ 등의 클래스 기반 언어와 같이 `class` 키워드를 사용하여 구체적인 객체로 인스턴스화하는 추상적인 클래스 정의를 만드는 대신, ActionScript 1.0과 같은 프로토타입 기반 언어에서는 기존 객체를 모델(프로토타입)로 사용하여 다른 객체를 만듭니다. 클래스 기반 언어의 객체는 자신의 템플릿 역할을 하는 클래스를 가리킬 수 있지만, 프로토타입 기반 언어의 객체는 자신의 템플릿 역할을 하는 프로토타입이라는 다른 객체를 대신 가리킵니다.

ActionScript 1.0에서 클래스를 만들려면 해당 클래스의 생성자 함수를 정의해야 합니다. ActionScript에서는 함수가 추상적인 정의가 아닌 실제 객체입니다. 작성된 생성자 함수는 해당 클래스 인스턴스의 프로토타입 객체 역할을 합니다. 다음 코드에서는 Shape라는 클래스를 만들고 기본적으로 `true`로 설정되는 `visible`이라는 속성 하나를 정의합니다.

```
// 기본 클래스
function Shape() {}
// visible이라는 속성을 만듭니다.
Shape.prototype.visible = true;
```

이 생성자 함수는 다음과 같이 `new` 연산자로 인스턴스화할 수 있는 Shape 클래스를 정의합니다.

```
myShape = new Shape();
```

Shape() 생성자 함수는 Shape 클래스 인스턴스의 프로토타입 역할을 할 뿐만 아니라 Shape의 하위 클래스(Shape 클래스를 확장하는 다른 클래스)의 프로토타입 역할도 합니다.

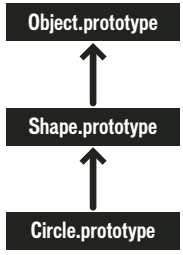
Shape 클래스의 하위 클래스를 만들려면 두 단계를 거쳐야 합니다. 우선 다음과 같이 클래스의 생성자 함수를 정의하여 클래스를 만듭니다.

```
// 자식 클래스
function Circle(id, radius)
{
    this.id = id;
    this.radius = radius;
}
```

다음으로 new 연산자를 사용하여 Shape 클래스가 Circle 클래스의 프로토타입임을 선언합니다. 기본적으로 작성되는 모든 클래스에는 Object 클래스가 프로토타입으로 사용됩니다. 즉, Circle.prototype에는 현재 일반 객체(Object 클래스의 인스턴스)가 들어 있습니다. Circle의 프로토타입을 Object 대신 Shape로 지정하려면 다음 코드를 사용하여 Circle.prototype에 일반 객체 대신 Shape 객체가 포함되도록 값을 변경합니다.

```
// Circle을 Shape의 하위 클래스로 만듭니다.
Circle.prototype = new Shape();
```

Shape 클래스와 Circle 클래스는 이제 상속 관계로 연결되었으며 이러한 관계를 일반적으로 *프로토타입 체인*이라고 합니다. 다음 그림은 프로토타입 체인 관계를 보여 줍니다.



모든 프로토타입 체인 끝에 있는 기본 클래스는 Object 클래스입니다. Object 클래스에는 Object.prototype이라는 정적 속성이 들어 있으며 이 속성은 ActionScript 1.0에서 만드는 모든 객체의 기본 프로토타입 객체를 가리킵니다. 예제 프로토타입 체인에서 다음에 있는 객체는 Shape 객체입니다. 이는 Shape.prototype 속성을 명시적으로 설정하지 않았으므로 일반 객체(Object 클래스의 인스턴스)가 계속 들어 있기 때문입니다. 이 체인의 마지막 링크는 해당 프로토타입인 Shape 클래스에 연결되어 있는 Circle 클래스입니다. Circle.prototype 속성에는 Shape 객체가 들어 있습니다.

다음 예제와 같이 Circle 클래스의 인스턴스를 만들면 Circle 클래스의 프로토타입 체인이 인스턴스로 상속됩니다.

```
// Circle 클래스의 인스턴스를 만듭니다.
myCircle = new Circle();
```

Shape 클래스의 멤버로 visible이라는 속성을 만들었습니다. 예제에서는 visible 속성이 myCircle 객체에 포함되어 있지 않고 Shape 객체의 멤버일 뿐이지만, 다음 코드 행을 실행하면 true가 산출됩니다.

```
trace(myCircle.visible); // 출력 : true
```

Flash Player는 프로토타입 체인을 확인하여 myCircle 객체에 visible 속성이 상속됨을 인식할 수 있습니다. 이 코드를 실행하면 Flash Player는 우선 myCircle 객체의 속성 중에서 이름이 visible인 속성을 검색하지만 이러한 속성을 찾을 수 없습니다. Flash Player는 다음으로 Circle.prototype 객체에서 속성을 검색하지만 visible이라는 속성을 여전히 찾을 수 없습니다. 프로토타입 체인을 거슬러 올라가면서 Flash Player는 Shape.prototype 객체에 정의된 visible 속성을 찾게 되고 이 속성의 값을 출력합니다.

이 단원에서는 이해를 돕기 위해 프로토타입 체인과 관련된 자세한 세부 내용이 상당 부분 생략되었으며, ActionScript 3.0 객체 모델을 이해하는 데 필요한 수준의 정보를 제공하고 있습니다.

ActionScript 2.0

ActionScript 2.0에는 class, extends, public 및 private 등의 키워드가 새로 도입되었으며, 이를 통해 Java 및 C++ 등의 클래스 기반 언어를 사용하는 작업자에게 익숙한 방식으로 클래스를 정의할 수 있습니다. 단, ActionScript 1.0과 ActionScript 2.0 사이에서 기본 상속 메커니즘은 변경되지 않았습니다. ActionScript 2.0에는 클래스를 정의하는 구문이 새로 추가되었을 뿐입니다. 프로토타입 체인은 두 가지 언어 버전에서 동일한 방식으로 작동합니다.

다음 인용 코드와 같이 ActionScript 2.0에 새로 도입된 구문을 통해 많은 프로그래머에게 친숙한 방식으로 클래스를 정의할 수 있습니다.

```
// 기본 클래스
class Shape
{
    var visible:Boolean = true;
}
```

ActionScript 2.0에는 컴파일 타임 유형 확인에 사용되는 유형 약어도 도입되었습니다. 이를 통해 이전 예제의 visible 속성에 부울 값만 포함되도록 선언할 수 있습니다. 또한 새로운 extends 키워드를 통해 하위 클래스를 간편하게 만들 수 있습니다. 다음 예제에서는 ActionScript 1.0의 경우 두 단계가 필요한 작업을 extends 키워드를 사용하여 한 단계로 수행합니다.

```
// 자식 클래스
class Circle extends Shape
{
  var id:Number;
  var radius:Number;
  function Circle(id, radius)
  {
```

```
        this.id = id;
        this.radius = radius;
    }
}
```

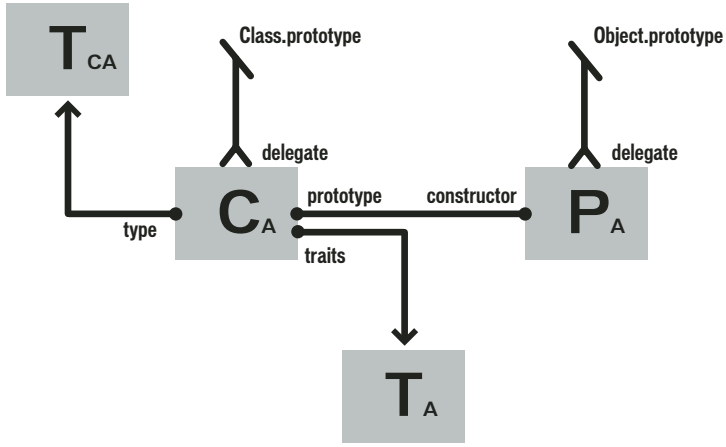
생성자는 이제 클래스 정의의 일부로 선언되며 클래스 속성인 `id` 및 `radius`도 명시적으로 선언해야 합니다.

ActionScript 2.0에는 인터페이스를 정의하는 기능이 추가되었으며, 이를 통해 객체 간 통신을 위해 정식으로 정의된 프로토콜을 사용하여 객체 지향 프로그램을 개선할 수 있습니다.

ActionScript 3.0 클래스 객체

일반적으로 Java 및 C++와 관련된 객체 지향 프로그래밍 패러다임에서는 클래스를 사용하여 객체 유형을 정의합니다. 또한 이러한 패러다임을 채택한 프로그래밍 언어에서는 클래스를 사용하여 클래스가 정의하는 데이터 유형의 인스턴스를 만듭니다. ActionScript에서도 이러한 두 가지 용도로 클래스를 사용하지만, ActionScript는 프로토타입 기반 언어로 시작되었으므로 독특한 특성이 있습니다. ActionScript에서는 각 클래스 정의마다 비헤이비어와 상태를 공유할 수 있는 특수 클래스 객체를 만듭니다. 그러나 대부분의 ActionScript 프로그래머는 이 특징으로 인해 코딩에 실질적인 영향을 받지는 않습니다. ActionScript 3.0은 이러한 특수 클래스 객체를 사용하거나 이해하지 않고도 복잡한 객체 지향 ActionScript 응용 프로그램을 만들 수 있도록 설계되었습니다. 이 단원에서는 클래스 객체를 활용하려는 고급 프로그래머를 위해 이 문제를 심도 있게 설명합니다.

다음 그림에는 `class A {}` 문으로 정의된 A라는 간단한 클래스를 나타내는 클래스 객체의 구조가 표현되어 있습니다.



그림의 각 사각형은 객체를 나타냅니다. 그림의 각 객체에는 A라는 아래 첨자가 있으며 이 문자는 클래스 A에 속함을 나타냅니다. 클래스 객체(C_A)에는 중요한 몇 가지 다른 객체를 가리키는 참조가 들어 있습니다. 인스턴스 traits 객체(T_A)에는 클래스 정의 내에 정의된 인스턴스 속성이 저장됩니다. 클래스 traits 객체(T_{CA})는 클래스의 내부 유형을 나타내며 이 객체에는 클래스에 정의된 정적 속성이 저장됩니다. 여기에서 아래 첨자 C는 "클래스"를 나타냅니다. 프로토타입 객체(P_A)는 항상 `constructor` 속성을 통해 자신이 원래 연결된 클래스 객체를 참조합니다.

traits 객체

ActionScript 3.0에 새로 도입된 traits 객체는 성능을 고려하여 구현되었습니다. 이전 버전의 ActionScript에서는 이름을 조회할 때 Flash Player에서 프로토타입 체인을 확인했으므로 시간이 오래 걸렸습니다. ActionScript 3.0에서는 상속된 속성이 수퍼 클래스에서 하위 클래스의 traits 객체로 복사되므로 이름을 훨씬 빠르고 효율적으로 조회할 수 있습니다.

traits 객체는 프로그래머 코드에서 직접 액세스할 수 없지만 성능 및 메모리 사용을 개선해 줍니다. traits 객체는 AVM2에 클래스의 레이아웃과 내용에 대한 자세한 정보를 제공합니다. AVM2에서는 이러한 정보를 바탕으로 시간이 걸리는 이름 조회를 수행하지 않고 직접 속성에 액세스하거나 메서드를 호출하는 기계어 명령을 생성할 수 있으므로 실행 시간이 크게 줄어듭니다.

traits 객체가 도입됨에 따라 객체가 점유하는 메모리가 이전 버전의 ActionScript보다 크게 줄어듭니다. 예를 들어, 클래스가 동적으로 선언되지 않아 봉인된 경우에는 클래스의 인스턴스에 동적으로 추가된 속성을 위한 해시 테이블이 필요하지 않으므로, 인스턴스에는 traits 객체에 대한 포인터 및 클래스에 정의된 고정 속성을 위한 슬롯 몇 개만 있으면 됩니다. 따라서 ActionScript 2.0에서는 100바이트의 메모리가 필요한 객체가 ActionScript 3.0에서는 20바이트만 필요할 수도 있습니다.

참고 traits 객체는 내부적으로 구현되는 세부 사항이며, 이후 버전의 ActionScript에서는 변경되거나 없어질 수도 있습니다.

프로토타입 객체

모든 ActionScript 클래스 객체에는 클래스의 프로토타입 객체를 참조하는 `prototype`이라는 속성이 있습니다. 프로토타입 객체는 ActionScript가 프로토타입 기반 언어로 시작되었기 때문에 남아 있는 항목입니다. 자세한 내용은 163페이지의 "ActionScript 1.0"을 참조하십시오.

prototype 속성은 읽기 전용이므로 다른 객체를 가리키도록 수정할 수 없습니다.
이는 ActionScript 이전 버전의 클래스 prototype 속성과 다른 점입니다. 이전 버전에서는 다른 클래스를 가리키도록 프로토타입을 다시 지정할 수 있었습니다. prototype 속성은 읽기 전용이지만, 이 속성이 참조하는 프로토타입 객체는 그렇지 않습니다. 즉, 프로토타입 객체에 새 속성을 추가할 수 있습니다. 프로토타입 객체에 추가된 속성은 클래스의 모든 인스턴스 사이에 공유됩니다.

이전 버전의 ActionScript에서 유일한 상속 메커니즘인 프로토타입 체인은 ActionScript 3.0에서 보조적인 역할만 합니다. 기본 상속 메커니즘인 고정 속성 상속은 traits 객체에 의해 내부적으로 처리됩니다. 고정된 속성은 클래스 정의의 일부로 정의된 변수 또는 메서드입니다. 고정 속성 상속은 class, extends 및 override 등의 키워드와 관련된 상속 메커니즘이므로 클래스 상속이라고도 합니다.

프로토타입 체인은 고정 속성 상속 대신 사용할 수 있는 더욱 동적인 상속 메커니즘을 제공합니다. 클래스를 정의할 때뿐만 아니라 런타임에도 클래스 객체의 prototype 속성을 통해 클래스의 프로토타입 객체에 속성을 추가할 수 있습니다. 그러나 컴파일러가 strict 모드로 설정되어 있으면 클래스를 dynamic 키워드로 선언하지 않은 경우에는 프로토타입 객체에 추가된 속성에 액세스하지 못할 수도 있습니다.

Object 클래스는 프로토타입 객체에 연결된 여러 속성을 가진 클래스의 전형입니다. Object 클래스의 toString() 및 valueOf() 메서드는 실제로는 Object 클래스의 프로토타입 객체의 속성에 지정된 함수입니다. 다음 예제에서는 이러한 메서드가 어떻게 선언될 수 있는지를 이론적 관점에서 보여 줍니다. 그러나 실제 구현은 세부 구현 사항에 따라 다릅니다.

```
public dynamic class Object
{
  prototype.toString = function()
  {
    // 명령문
  };
  prototype.valueOf = function()
  {
    // 명령문
  };
}
```

앞에서 언급했듯이 클래스 정의 외부에 있는 클래스의 프로토타입 객체에 속성을 연결할 수 있습니다. 예를 들어, toString() 메서드를 다음과 같이 Object 클래스 정의 외부에 정의할 수도 있습니다.

```
Object.prototype.toString = function()
{
  // 명령문
};
```

그러나 고정 속성 상속과는 달리 프로토타입 상속에서는 하위 클래스에서 메서드를 재정의하는 경우 override 키워드가 필요하지 않습니다. 예를 들어, Object 클래스의 하위 클래스에서 valueOf() 메서드를 재정의하려는 경우 세 가지 옵션 중 하나를 선택할 수 있습니다. 첫 번째 옵션은 클래스 정의 내에 있는 하위 클래스의 프로토타입 객체에 valueOf() 메서드를 정의하는 것입니다. 다음 코드에서는 Object의 하위 클래스 Foo를 만들고 클래스 정의의 일부로서 Foo의 프로토타입 객체에 valueOf() 메서드를 재정의합니다. 모든 클래스는 Object에서 상속되므로 extends 키워드는 사용하지 않아도 됩니다.

```
dynamic class Foo
{
  prototype.valueOf = function()
  {
    return "Instance of Foo";
  };
}
```

두 번째 옵션은 다음 코드에 나와 있듯이 클래스 정의 외부에 있는 Foo의 프로토타입 객체에 valueOf() 메서드를 정의하는 것입니다.

```
Foo.prototype.valueOf = function()
{
  return "Instance of Foo";
};
```

세 번째 옵션은 valueOf()라는 이름의 고정 속성을 Foo 클래스의 일부로서 정의하는 것입니다. 이 기법은 고정 속성 상속과 프로토타입 상속을 혼합한다는 점에서 다른 두 옵션과 차별화됩니다. Foo의 하위 클래스에서 valueOf() 메서드를 재정의할 때는 항상 override 키워드를 사용해야 합니다. 다음 코드에서는 Foo에 고정 속성으로 정의된 valueOf() 메서드를 보여 줍니다.

```
class Foo
{
  function valueOf():String
  {
    return "Instance of Foo";
  }
}
```

AS3 네임스페이스

고정 속성 상속과 프로토타입 상속이라는 별개의 상속 메커니즘이 공존함으로 인해 기본 클래스의 속성 및 메서드와 관련하여 흥미로운 호환성 문제가 발생합니다. ECMAScript, Edition 4 초안의 언어 사양과 호환성을 갖추려면 프로토타입 상속을 사용해야 합니다. 이는 기본 클래스의 속성 및 메서드가 해당 클래스의 프로토타입 객체에 정의되어야 함을 의미합니다. 반면에 Flash Player API와 호환성을 갖추려면 고정 속성 상속을 사용해야 합니다. 이는 `const`, `var` 및 `function` 키워드를 사용하여 기본 클래스의 속성 및 메서드를 클래스 정의에 정의해야 함을 의미합니다. 프로토타입 버전 대신 고정 속성을 사용하면 런타임 성능이 크게 향상되는 장점도 있습니다.

ActionScript 3.0에서는 기본 클래스에 대해 프로토타입 상속 및 고정 속성 상속을 모두 사용하여 이 문제를 해결합니다. 각 기본 클래스에는 속성 및 메서드 집합이 두 개씩 있습니다. 한 집합은 ECMAScript 사양과의 호환성을 위해 프로토타입 객체에 정의되어 있고 다른 집합은 Flash Player API와의 호환성을 위해 고정 속성 및 AS3 네임스페이스를 사용하여 정의되어 있습니다.

AS3 네임스페이스는 두 개의 속성 및 메서드 집합 중 하나를 선택할 때 사용할 수 있는 편리한 메커니즘을 제공합니다. AS3 네임스페이스를 사용하지 않으면 기본 클래스의 인스턴스에서는 기본 클래스의 프로토타입 객체에 정의된 속성 및 메서드를 상속합니다. AS3 네임스페이스를 사용하면 항상 프로토타입 속성 대신 고정 속성이 선택되므로 기본 클래스의 인스턴스에서는 AS3 버전을 상속합니다. 즉, 고정 속성을 사용할 수 있는 경우에는 언제나 동일한 이름의 프로토타입 속성 대신 고정 속성이 사용됩니다.

속성 또는 메서드를 선택하여 AS3 네임스페이스로 정규화하면 해당 속성 또는 메서드의 AS3 네임스페이스 버전을 사용할 수 있습니다. 예를 들어, 다음 코드에서는 `Array.pop()` 메서드의 AS3 버전이 사용됩니다.

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // 출력 : 1,2
```

또는 `use namespace` 지시문을 사용하여 코드 블록 내의 모든 정의에 대한 AS3 네임스페이스를 열 수 있습니다. 예를 들어 다음 코드에서는 `use namespace` 지시문을 사용하여 `pop()` 및 `push()` 메서드에 대한 AS3 네임스페이스를 엽니다.

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // 출력 : 1,2,5
```

ActionScript 3.0에서는 각 속성 집합에 대한 컴파일러 옵션을 제공하여 전체 프로그램에 AS3 네임스페이스를 적용할 수 있도록 합니다. `-as3` 컴파일러 옵션은 AS3 네임스페이스를 나타내며 `-es` 컴파일러 옵션은 프로토타입 상속 옵션을 나타냅니다. 여기서 `es`는 ECMAScript를 의미합니다. 전체 프로그램에 대해 AS3 네임스페이스를 열려면 `-as3` 컴파일러 옵션을 `true`로 설정하고 `-es` 컴파일러 옵션을 `false`로 설정합니다. 프로토타입 버전을 사용하려면 컴파일러 옵션의 값을 반대로 설정하면 됩니다. Adobe Flex Builder 2 및 Adobe Flash CS3 Professional에서 기본 컴파일러 설정은 `-as3 = true` 및 `-es = false`입니다.

기본 클래스를 확장하고 메서드를 재정의하려면 재정의된 메서드를 선언하는 방법에 AS3 네임스페이스가 어떻게 영향을 미칠 수 있는지 이해해야 합니다. AS3 네임스페이스를 사용하고 있는 경우에 기본 클래스의 메서드를 재정의하려면 `override` 특성과 함께 AS3 네임스페이스를 사용해야 합니다. AS3 네임스페이스를 사용하지 않는 상태에서 기본 클래스 메서드를 하위 클래스에서 재정의하려면 AS3 네임스페이스나 `override` 키워드를 사용해서는 안됩니다.

예제: GeometricShapes

GeometricShapes 샘플 응용 프로그램에서는 ActionScript 3.0을 사용하여 얼마나 많은 객체 지향 개념과 기능을 적용할 수 있는지 보여 줍니다.

- 클래스 정의
- 클래스 확장
- 다형성 및 `override` 키워드
- 정의, 확장 및 인터페이스 구현

이 예제에는 클래스 인스턴스를 생성하는 "팩토리 메서드"가 나와 있으며 반환값을 인터페이스의 인스턴스로 선언하고 반환된 객체를 일반적인 방식으로 사용하는 방법도 보여 줍니다.

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. GeometricShapes 응용 프로그램 파일은 Samples/GeometricShapes 폴더에서 찾을 수 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
GeometricShapes.mxml 또는 GeometricShapes.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/geometricshapes/IGeometricShape.as	모든 기하학적 모양 클래스에서 구현될 메서드를 정의하는 기본 인터페이스입니다.

파일	설명
com/example/programmingas3/geometricshapes/IPolygon.as	변이 여러 개인 GeometricShapes 응용 프로그램 클래스에서 구현될 메서드를 정의하는 인터페이스입니다.
com/example/programmingas3/geometricshapes/RegularPolygon.as	길이가 같은 여러 변이 중심을 기준으로 대칭을 이루는 기하학적 형태 유형입니다.
com/example/programmingas3/geometricshapes/Circle.as	원을 정의하는 기하학적 모양 유형입니다.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	모든 변의 길이가 같은 삼각형을 정의하는 RegularPolygon의 하위 클래스입니다.
com/example/programmingas3/geometricshapes/Square.as	네 변의 길이가 모두 같은 사각형을 정의하는 RegularPolygon의 하위 클래스입니다.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	지정된 모양 유형과 크기로 모양을 만드는 factory 메서드가 포함된 클래스입니다.

GeometricShapes 클래스 정의

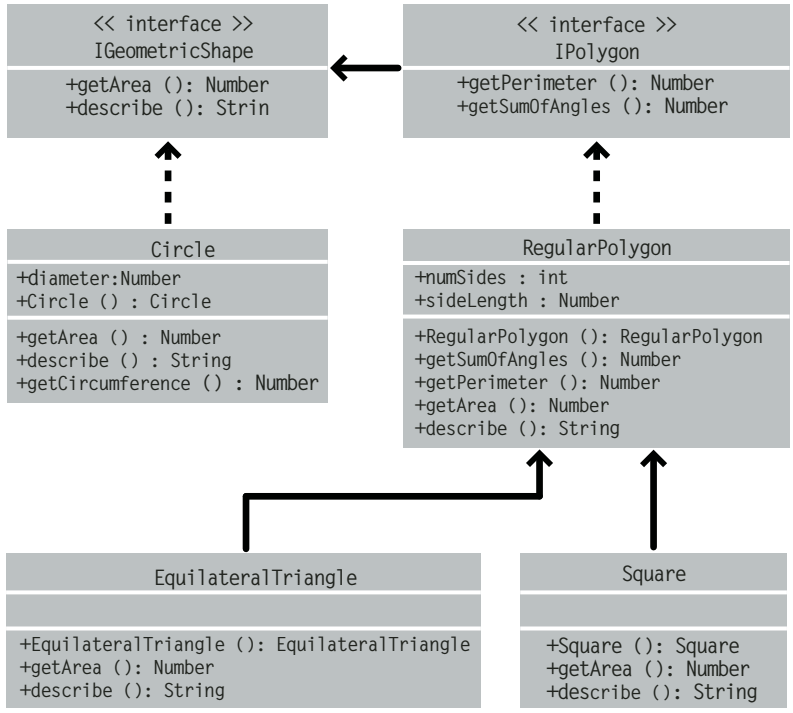
GeometricShapes 응용 프로그램에서는 사용자가 기하학적 형태의 유형 및 크기를 지정하도록 합니다. 그런 다음 사용자에게 형태에 대해 설명하고 면적 및 둘레를 알려 줍니다.

이 응용 프로그램의 사용자 인터페이스는 간단하게 구성되어 있습니다. 형태의 유형을 선택하고 크기를 설정하고 설명을 표시하는 데 사용하는 컨트롤 몇 개로 구성됩니다. 이 응용 프로그램에서 가장 흥미로운 부분은 표면에 드러나지 않는 클래스 및 인터페이스의 구조에 있습니다.

이 응용 프로그램에서는 기하학적 형태를 다루지만 그래픽으로 표시하지는 않습니다. 이 응용 프로그램에서는 제12장의 397페이지의 “예제: SpriteArranger”에서 재사용되는 클래스 및 인터페이스로 구성된 작은 라이브러리를 제공합니다. SpriteArranger 예제에서는 이 장의 GeometricShapes 응용 프로그램에서 제공하는 클래스 프레임워크에 기반하여 형태를 그래픽으로 표시하고 사용자가 이를 조작할 수 있게 합니다.

이 예제에서 기하학적 형태를 정의하는 클래스 및 인터페이스는 UML(Unified Modeling Language)을 사용하여 다음 다이어그램과 같이 나타낼 수 있습니다.

GeometricShapes 클래스 예제



인터페이스를 사용하여 공통 비헤이비어 정의

이 GeometricShapes 응용 프로그램에서는 세 가지 유형의 형태 즉, 원, 사각형 및 등변 삼각형을 다룹니다. GeometricShapes의 클래스 구조는 매우 작은 인터페이스인 IGeometricShape에서 시작됩니다. 이 인터페이스에는 세 가지 유형의 형태 모두에 공통적인 메서드가 나열되어 있습니다.

```
package com.example.programmingas3.geometricshapes
{
  public interface IGeometricShape
  {
    function getArea():Number;
    function describe():String;
  }
}
```

이 인터페이스에는 두 개의 메서드가 정의되어 있습니다. `getArea()` 메서드는 형태의 면적을 계산하여 반환하며 `describe()` 메서드는 문자열을 결합하여 형태의 속성을 설명하는 텍스트를 만듭니다.

각 모양의 둘레도 확인하고 싶은 경우를 가정해 봅니다. 원의 둘레는 원주라고 하며 고유한 계산 방법이 있으므로 비헤이비어가 삼각형 또는 사각형과 달라지게 됩니다. 그러나 삼각형, 사각형 및 기타 다각형 간에는 충분한 유사성이 있으므로 이들을 위한 새로운 클래스 인터페이스 즉, IPolygon을 정의하는 것은 의미가 있습니다. 또한 IPolygon 인터페이스는 여기에 나와 있듯이 상당히 간단하게 정의됩니다.

```
package com.example.programmingas3.geometricshapes
{
  public interface IPolygon extends IGeometricShape
  {
    function getPerimeter():Number;
    function getSumOfAngles():Number;
  }
}
```

이 인터페이스에는 모든 다각형에 공통적인 두 개의 메서드가 정의되어 있습니다. `getPerimeter()` 메서드는 모든 변의 길이를 합산하여 측정하며 `getSumOfAngles()` 메서드는 내부 각도를 모두 더합니다.

IPolygon 인터페이스는 IGeometricShape 인터페이스를 확장하므로 IPolygon 인터페이스를 구현하는 모든 클래스에서는 IGeometricShape 인터페이스와 IPolygon 인터페이스에 두 개씩 정의되어 있는 네 개의 메서드를 모두 선언해야 합니다.

형태 클래스 정의

형태 유형별로 공통적인 메서드가 발견되면 각각의 형태 클래스를 정의할 수 있습니다. 구현해야 하는 메서드의 수를 기준으로 할 때 가장 간단한 형태는 다음에 나와 있듯이 Circle 클래스입니다.

```
package com.example.programmingas3.geometricshapes
{
  public class Circle implements IGeometricShape
  {
    public var diameter:Number;

    public function Circle(diam:Number = 100):void
    {
      this.diameter = diam;
    }

    public function getArea():Number
    {
      // 수식은 Pi * radius * radius 입니다 .
      var radius:Number = diameter / 2;
      return Math.PI * radius * radius;
    }

    public function getCircumference():Number
    {
      // 수식은 Pi * diameter 입니다 .
      return Math.PI * diameter;
    }

    public function describe():String
    {
      var desc:String = "This shape is a Circle.\n";
      desc += "Its diameter is " + diameter + " pixels.\n";
      desc += "Its area is " + getArea() + ".\n";
      desc += "Its circumference is " + getCircumference() + ".\n";
      return desc;
    }
  }
}
```

Circle 클래스는 IGeometricShape 인터페이스를 구현하므로 `getArea()` 메서드 및 `describe()` 메서드에 대한 코드가 작성되어야 합니다. 또한 이 클래스에는 `getCircumference()`라는 고유한 메서드가 정의되어 있습니다. Circle 클래스에는 다른 다각형 클래스에서는 찾을 수 없는 `diameter`라는 속성도 선언되어 있습니다.

다른 두 유형의 형태 즉, 정사각형과 등변 삼각형은 몇 가지 추가적인 공통점이 있습니다. 예를 들어, 두 형태 모두 변의 길이가 동일하며 일반적인 공식을 사용하여 둘레 및 내부 각도의 합을 계산할 수 있습니다. 사실 이러한 일반적인 공식은 앞으로 정의해야 하는 다른 모든 일반 다각형에도 적용됩니다.

RegularPolygon 클래스는 이후에 Square 클래스 및 EquilateralTriangle 클래스의 수퍼 클래스가 됩니다. 수퍼 클래스를 사용하면 한 곳에 공통적인 메서드를 정의할 수 있으므로 이러한 메서드를 각 하위 클래스에서 별도로 정의하지 않아도 됩니다. RegularPolygon 클래스의 코드가 다음에 나와 있습니다.

```
package com.example.programmingas3.geometricshapes
{
  public class RegularPolygon implements IPolygon
  {
    public var numSides:int;
    public var sideLength:Number;

    public function RegularPolygon(len:Number = 100, sides:int = 3):void
    {
      this.sideLength = len;
      this.numSides = sides;
    }

    public function getArea():Number
    {
      // 이 메서드는 하위 클래스에서 재정의됩니다 .
      return 0;
    }

    public function getPerimeter():Number
    {
      return sideLength * numSides;
    }

    public function getSumOfAngles():Number
    {
      if (numSides >= 3)
      {
        return ((numSides - 2) * 180);
      }
      else
      {
        return 0;
      }
    }

    public function describe():String
    {
      var desc:String = "Each side is " + sideLength + " pixels long.\n";
```

```
        desc += "Its area is " + getArea() + " pixels square.\n";
        desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
        desc += "The sum of all interior angles in this shape is " +
    getSumOfAngles() + " degrees.\n";
        return desc;
      }
    }
}
```

RegularPolygon 클래스에서는 먼저 모든 일반 다각형에 공통적인 두 가지 속성 즉, `sideLength` 속성(각 변의 길이) 및 `numSides` 속성(변의 수)을 선언합니다.

RegularPolygon 클래스에서는 IPolygon 인터페이스를 구현하며 이 인터페이스의 메서드 네 개를 모두 선언합니다. 이러한 메서드 중 두 개 즉, `getPerimeter()` 및 `getSumOfAngles()` 메서드는 일반적인 공식을 사용하여 구현합니다.

`getArea()` 메서드에 사용되는 공식은 형태에 따라 다르므로 이 메서드의 기본 클래스 버전에는 하위 클래스 메서드에 상속될 수 있는 공통적인 논리를 포함할 수 없습니다. 대신 간단히 기본값 0을 반환하여 면적이 계산되지 않았음을 나타냅니다. 각 형태의 면적을 올바르게 계산하려면 RegularPolygon 클래스의 하위 클래스에서 `getArea()` 메서드를 재정의해야 합니다.

EquilateralTriangle 클래스의 다음 코드에서는 `getArea()` 메서드를 재정의하는 방법을 보여 줍니다.

```
package com.example.programmingas3.geometricshapes
{
  public class EquilateralTriangle extends RegularPolygon
  {
    public function EquilateralTriangle(len:Number = 100):void
    {
      super(len, 3);
    }

    public override function getArea():Number
    {
        // 수식은 ((sideLength 제곱 ) * (3 의 제곱근 )) / 4 입니다 .
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
    }

    public override function describe():String
    {
         /* 해당 형태의 이름으로 시작하고 나머지 설명 작업은 RegularPolygon
         수퍼 클래스에 위임합니다 . */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
    }
  }
}
```

override 키워드는 EquilateralTriangle.getArea() 메서드로 RegularPolygon 수퍼 클래스의 getArea() 메서드를 재정의하려는 의도를 나타냅니다. EquilateralTriangle.getArea() 메서드가 호출되면 위 코드의 공식을 사용하여 면적을 계산하며 RegularPolygon.getArea() 메서드의 코드는 결코 실행되지 않습니다.

한편 EquilateralTriangle 클래스에 getPerimeter() 메서드의 자체 버전은 정의되어 있지 않습니다. EquilateralTriangle.getPerimeter() 메서드가 호출되면 이 호출은 상속 체인을 따라 위로 전달되어 RegularPolygon 클래스의 getPerimeter() 메서드의 코드가 실행됩니다.

EquilateralTriangle() 생성자에서는 super() 문을 사용하여 수퍼 클래스의 RegularPolygon() 생성자를 명시적으로 호출합니다. 두 생성자의 매개 변수 집합이 동일한 경우에는 EquilateralTriangle() 생성자를 완전히 생략할 수 있으며 이렇게 하면 RegularPolygon() 생성자가 대신 실행됩니다. 그러나 RegularPolygon() 생성자에는 numSides라는 추가적인 매개 변수가 필요합니다. 따라서 EquilateralTriangle() 생성자에서는 super(len, 3)을 호출하여 len 입력 매개 변수와 값 3을 전달함으로써 삼각형의 변이 3개임을 알립니다.

describe() 메서드에서도 super() 문을 사용하여 describe() 메서드의 RegularPolygon 수퍼 클래스 버전을 호출하지만 위와는 다른 방법이 적용됩니다. EquilateralTriangle.describe() 메서드에서는 먼저 desc 문자열 변수를 형태 유형에 대한 명령문으로 설정합니다. 그런 다음 super.describe()를 호출하여 RegularPolygon.describe() 메서드의 결과를 구하고 이 결과를 desc 문자열에 추가합니다.

Square 클래스는 여기서 자세히 설명되지는 않지만 생성자와 getArea() 및 describe() 메서드의 자체 구현을 제공한다는 점에서 EquilateralTriangle 클래스와 유사합니다.

다형성 및 팩토리 메서드

인터페이스 및 상속을 잘 활용하는 클래스의 집합은 여러 가지 흥미로운 방식으로 사용될 수 있습니다. 예를 들어, 지금까지 설명한 모든 형태 클래스는 IGeometricShape 인터페이스를 직접 구현하거나 이 인터페이스를 구현한 수퍼 클래스를 확장합니다. 따라서 변수를 IGeometricShape의 인스턴스로 정의하면 describe() 메서드를 호출하기 위해 해당 인스턴스가 실제로 Circle 클래스의 인스턴스인지 아니면 Square 클래스의 인스턴스인지 알아야 할 필요가 없습니다.

다음 코드에서는 이에 대한 예제를 보여 줍니다.

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

변수가 IGeometricShape 인터페이스의 인스턴스로 정의되어 있지만 기반 클래스는 Circle이므로 myShape.describe()를 호출하면 Circle.describe()가 실행됩니다.

이 예제에서는 다형성의 원리가 적용됨을 보여 줍니다. 정확히 동일한 메서드를 호출해도 메서드가 호출된 객체의 클래스에 따라 다른 코드가 실행되는 것을 확인할 수 있습니다.

GeometricShapes 응용 프로그램에서는 팩토리 메서드로 알려진 디자인 패턴의 단순화된 버전을 사용하여 이러한 종류의 인터페이스 기반 다형성을 적용합니다. *팩토리 메서드*라는 용어는 기본 데이터 유형이나 내용이 컨텍스트에 따라 달라질 수 있는 객체를 반환하는 함수를 가리킵니다.

여기에 나와 있는 GeometricShapeFactory 클래스에는 `createShape()`라는 이름의 팩토리 메서드가 정의되어 있습니다.

```
package com.example.programmingas3.geometricshapes
{
  public class GeometricShapeFactory
  {
    public static var currentShape:IGeometricShape;

    public static function createShape(shapeName:String,
                                       len:Number):IGeometricShape
    {
      switch (shapeName)
      {
        case "Triangle":
          return new EquilateralTriangle(len);

        case "Square":
          return new Square(len);

        case "Circle":
          return new Circle(len);
      }
      return null;
    }

    public static function describeShape(shapeType:String,
  shapeSize:Number):String
    {
      GeometricShapeFactory.currentShape =
        GeometricShapeFactory.createShape(shapeType, shapeSize);
      return GeometricShapeFactory.currentShape.describe();
    }
  }
}
```

`createShape()` 팩토리 메서드에서는 형태 하위 클래스의 생성자에서 생성되는 인스턴스의 세부 사항을 정의하도록 하는 한편 새 객체를 반환할 때는 응용 프로그램에서 더 일반적인 방식으로 이 객체를 처리할 수 있도록 IGeometricShape 인스턴스로 반환합니다.

이전 예제의 `describeShape()` 메서드는 응용 프로그램에서 팩토리 메서드를 사용하여 더 구체적인 객체에 대한 일반적인 참조를 얻을 수 있는 방법을 보여 줍니다. 응용 프로그램에서는 새로 생성된 Circle 객체에 대한 설명을 다음과 같이 확인할 수 있습니다.

```
GeometricShapeFactory.describeShape("Circle", 100);
```

그런 다음 `describeShape()` 메서드에서는 동일한 매개 변수를 사용하여 `createShape()` 팩토리 메서드를 호출하고 반환된 새 Circle 객체를 `currentShape`라는 정적 변수에 저장합니다. 이 변수의 유형은 IGeometricShape 객체입니다. 이어서 `currentShape` 객체에 대해 `describe()` 메서드를 호출하면 자동으로 `Circle.describe()` 메서드가 실행되고 이 메서드에서는 원에 대한 상세한 설명을 반환합니다.

샘플 응용 프로그램 개선

응용 프로그램을 개선하거나 변경하면 인터페이스 및 상속의 진정한 효과가 분명하게 드러납니다.

이 샘플 응용 프로그램에 새 형태로 5각형을 추가하려는 경우를 가정해 봅니다. RegularPolygon 클래스를 확장하고 `getArea()` 및 `describe()` 메서드의 자체 버전을 정의하는 Pentagon 클래스를 새로 만듭니다. 그런 다음 응용 프로그램의 사용자 인터페이스에 있는 콤보 상자에 Pentagon 옵션을 새로 추가합니다. 더 이상의 작업은 필요하지 않습니다. Pentagon 클래스에서는 상속을 통해 RegularPolygon 클래스의 `getPerimeter()` 메서드 및 `getSumOfAngles()` 메서드의 기능을 자동적으로 사용합니다. Pentagon 클래스는 IGeometricShape 인터페이스를 구현한 클래스에서 상속되었으므로 Pentagon 인스턴스를 IGeometricShape 인터페이스로도 취급할 수 있습니다. 따라서 GeometricShapeFactory 클래스에서 메서드를 변경할 필요가 전혀 없으므로 필요한 경우 새 유형의 형태를 훨씬 쉽게 추가할 수 있습니다.

Pentagon 클래스를 Geometric Shapes 예제에 추가하는 연습을 해 보십시오. 이 연습을 통해 응용 프로그램에 새 기능을 추가할 때의 작업 로드가 인터페이스와 상속으로 인해 얼마나 줄어들 수 있는지 확인하십시오.

제 5 장

날짜 및 시간을 사용한 작업

5

시간은 소프트웨어 응용 프로그램에서 매우 중요한 요소입니다. ActionScript 3.0에서는 달력 날짜, 시간 및 시간 간격을 관리할 수 있는 강력한 도구를 제공합니다. 두 개의 기본 클래스인 flash.utils 패키지의 Date 클래스 및 새로운 Timer 클래스에서 대부분의 시간 기능을 제공합니다.

목차

날짜 및 시간의 기초

날짜 및 시간 작업 소개

날짜와 시간은 ActionScript 프로그램에서 사용되는 일반적인 정보 유형입니다. 예를 들어, 오늘이 무슨 요일인지 알고 싶거나 다른 여러 화면 중 특정 화면에서 사용자가 시간을 얼마나 소비하는지를 측정할 수 있습니다. ActionScript에서는 Date 클래스를 사용하여 날짜 및 시간 정보를 비롯한 해당 시점을 나타낼 수 있습니다. Date 인스턴스 내에는 년, 월, 일, 요일, 시간, 분, 초, 밀리초 및 시간대를 포함하여 개별 날짜 및 시간 단위에 대한 값이 있습니다. ActionScript에 포함된 Timer 클래스를 사용하면 특정 지연 시간 후 또는 반복된 간격으로 작업을 수행하는 등, 고급 기능을 활용할 수 있습니다.

일반적인 날짜 및 시간 작업

이 장에서는 날짜 및 시간 정보와 관련된 다음과 같은 일반 작업에 대해 설명합니다.

- Date 객체 다루기
- 현재 날짜 및 시간 가져오기
- 개별 날짜 및 시간 단위(일, 년, 시간, 분 등) 액세스
- 날짜 및 시간 계산 수행
- 시간대 간 변환
- 반복 액션 수행
- 설정된 시간 간격 후 액션 수행

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- UTC 시간: Universal Time Coordinated의 약자로, "0시" 참조 시간대입니다. 다른 모든 시간대는 UTC 시간에 대한 전후 시간으로 정의됩니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플은 주로 Date 객체를 다루므로, 예제를 테스트하려면 텍스트 필드 인스턴스의 값을 스테이지에 기록하거나 `trace()` 함수를 사용하여 [출력] 패널에 값을 출력함으로써 해당 예제에 사용되는 변수 값을 표시해야 합니다. 이러한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

달력 날짜 및 시간 관리

ActionScript 3.0의 달력 날짜 및 시간 관리 함수는 최상위 Date 클래스에 집중되어 있습니다. Date 클래스에는 UTC(Coordinated Universal Time) 또는 시간대별 현지 시간에서 날짜 및 시간을 처리할 수 있는 메서드 및 속성이 들어 있습니다. UTC는 기본적으로 그리니치 표준시(GMT)와 동일한 표준 시간 정의입니다.

Date 객체 생성

Date 클래스는 모든 기본 클래스 중에서 가장 다양한 생성자 메서드를 제공하는 클래스입니다. 이 메서드는 네 가지 방법으로 호출할 수 있습니다.

첫 번째로, 매개 변수가 없는 경우 `Date()` 생성자에서 사용자 시간대의 현지 시간을 기준으로 현재 날짜와 시간을 포함하는 Date 객체를 반환합니다. 예를 들면 다음과 같습니다.

```
var now:Date = new Date();
```

두 번째로, 단일 숫자 매개 변수가 지정된 경우 `Date()` 생성자는 이를 1970년 1월 1일 이후로 경과된 밀리초로 처리하여 해당하는 Date 객체를 반환합니다. 즉, 전달된 밀리초 값은 표준시(UTC) 기준 1970년 1월 1일 이후로 경과된 밀리초로 처리됩니다. 하지만 Date 객체는 사용자가 UTC 특정 메서드를 사용하여 검색하고 표시하지 않는 한 현지 시간대로 값을 표시합니다. 단일 밀리초 매개 변수를 사용하여 새로운 Date 객체를 생성하는 경우 현지 시간과 표준시(UTC) 사이의 시차를 고려해야 합니다. 다음 명령문은 표준시(UTC) 기준 1970년 1월 1일 자정으로 설정된 Date 객체를 생성합니다.

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// 시작 날짜 1/1/1970 에서 하루 지난 Date 를 구합니다 .
var startTime:Date = new Date(millisecondsPerDay);
```

세 번째로, 여러 숫자 매개 변수를 `Date()` 생성자로 전달할 수 있습니다. 생성자는 전달된 매개 변수를 각각 연, 월, 일, 시, 분, 초 및 밀리초로 처리하고 해당하는 Date 객체를 반환합니다. 이러한 입력 매개 변수는 표준시(UTC)가 아니라 현지 시간으로 간주됩니다. 다음 명령문은 현지 시간 2000년 1월 1일 자정으로 설정된 Date 객체를 가져옵니다.

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

네 번째로, 단일 문자열 매개 변수를 `Date()` 생성자로 전달할 수 있습니다. 생성자는 해당 문자열을 날짜 및 시간 구성 요소로 파싱한 다음 해당하는 Date 객체를 반환합니다. 이 방법을 사용하는 경우에는 `try..catch` 블록에 `Date()` 생성자를 포함하여 모든 파싱 오류를 트랩하는 것이 좋습니다. `Date()` 생성자는 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에 나열된 여러 문자열 형식을 허용합니다. 다음 명령문은 문자열 값을 사용하여 새 Date 객체를 초기화합니다.

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Date() 생성자가 문자열 매개 변수를 성공적으로 파싱하지 못하는 경우에도 예외가 발생하지 않습니다. 하지만 결과 Date 객체에 잘못된 날짜 값이 포함됩니다.

시간 단위 값 가져오기

Date 클래스의 속성 또는 메서드를 사용하여 Date 객체 내의 여러 시간 단위 값을 추출할 수 있습니다. 다음 각 속성을 사용하여 Date 객체에서 시간 단위 값을 가져올 수 있습니다.

- fullYear 속성
- month 속성(1월의 0에서 12월의 11까지 숫자 형식 사용)
- date 속성(1부터 31까지 범위의 달력 날짜 사용)
- day 속성(일요일에 0 등 요일에 숫자 형식 사용)
- hours 속성(0 ~ 23 사용)
- minutes 속성
- seconds 속성
- milliseconds 속성

실제로 Date 클래스를 사용하면 여러 가지 방법으로 이 값을 가져올 수 있습니다. 예를 들어, 다음 네 가지 방법으로 Date 객체의 월 값을 가져올 수 있습니다.

- month 속성
- getMonth() 메서드
- monthUTC 속성
- getMonthUTC() 메서드

기본적으로 네 방법 모두 효율적이기 때문에 응용 프로그램에 따라 가장 적합한 방법을 사용하면 됩니다.

나열된 속성은 모두 총 날짜 값의 구성 요소를 나타냅니다. 예를 들어, milliseconds 속성은 999보다 클 수 없으며, 이는 값이 1000에 도달하면 초 값이 1 증가하고 milliseconds 속성이 0으로 재설정되기 때문입니다.

표준시 기준 1970년 1월 1일 이후로 경과된 밀리초로 Date 객체 값을 얻으려면 getTime() 메서드를 사용할 수 있습니다. 이에 대응되는 setTime() 메서드를 사용하면 표준시 기준 1970년 1월 1일 이후로 경과된 밀리초를 사용하여 기존 Date 객체를 변경할 수 있습니다.

날짜 및 시간 계산 수행

Date 클래스를 사용하여 날짜 및 시간에 대해 더하기 및 빼기를 수행할 수 있습니다. 날짜 값은 내부적으로 밀리초로 단위로 보관되므로 Date 객체에서 날짜 값을 더하거나 빼기 전에 값을 밀리초로 변환해야 합니다.

응용 프로그램에서 날짜 및 시간 계산을 많이 수행하는 경우에는 다음과 같이 상수를 생성하여 밀리초로 환산된 공통 시간 단위 값을 보관하는 것이 유용합니다.

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

표준 시간 단위를 사용하여 날짜 계산을 편리하게 수행할 수 있습니다. 다음 코드는 `getTime()` 및 `setTime()` 메서드를 사용하여 날짜 값을 현재 시간부터 한 시간으로 설정합니다.

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

날짜 값을 설정하는 또 다른 방법은 단일 밀리초 매개 변수를 사용하여 새 Date 객체를 생성하는 것입니다. 예를 들어, 다음 코드는 한 날짜에 30일을 더하여 다른 날짜를 계산합니다.

```
// 송장 날짜를 오늘 날짜로 설정합니다.
var invoiceDate:Date = new Date();

// 30일을 더해 만기일을 구합니다.
var dueDate:Date = new Date(invoiceDate.getTime() + (30 *
  millisecondsPerDay));
```

그런 다음 `millisecondsPerDay` 상수에 30을 곱해 30일의 시간을 나타내고 그 결과를 `invoiceDate` 값에 더하여 `dueDate` 값을 설정합니다.

시간대 간 변환

날짜 및 시간 계산은 날짜를 한 시간대에서 다른 시간대로 변환하고 싶은 경우에도 유용합니다. `getTimezoneOffset()` 메서드도 마찬가지이며, 이 메서드는 Date 객체의 시간대와 표준시의 차이를 분 값으로 반환합니다. 분 값으로 반환하는 이유는 일부 시간대가 시간 간격으로 시차가 설정되지 않고 옆 시간대에서 30분 차이가 나기도 하기 때문입니다.

다음 예제에서는 시간대 오프셋을 사용하여 현지 시간에서 표준시로 날짜를 변환합니다. 다음과 같이 먼저 밀리초 단위로 시간대 값을 계산한 다음 해당 값만큼 Date 값을 조정합니다.

```
// 현지 시간의 Date를 만듭니다.
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// 시간대 오프셋을 더하거나 빼서 Date를 UTC로 변환합니다.
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

시간 간격 제어

Adobe Flash CS3 Professional을 사용하여 응용 프로그램을 개발할 때는 응용 프로그램의 일정한 프레임 단위 진행을 가능하게 하는 타임라인을 사용할 수 있습니다. 하지만 ActionScript 프로젝트만 진행하는 경우에는 다른 시간 메커니즘을 사용해야 합니다.

루프와 타이머

일부 프로그래밍 언어에서는 `for` 또는 `do..while` 등의 루프 문을 사용하여 고유한 시간 체계를 만들어야 합니다.

일반적으로 루프 명령문은 로컬 시스템 성능에 따라 실행 속도가 달라지므로, 이에 따라 일부 시스템에서는 응용 프로그램이 빠르게 실행되지만 다른 시스템에서는 느리게 실행될 수 있습니다. 응용 프로그램에 일정한 시간 간격이 필요하면 실제 달력이나 시계를 연결해야 합니다. 게임, 애니메이션, 실시간 컨트롤러 등의 많은 응용 프로그램에는 시스템 모두에 일관되게 적용되는 규칙적인 시간 기반 티킹(ticking) 메커니즘이 필요합니다.

ActionScript 3.0 Timer 클래스는 매우 강력한 솔루션을 제공합니다. ActionScript 3.0 이벤트 모델을 사용하여, Timer 클래스는 지정된 시간 간격에 도달할 때마다 타이머 이벤트를 전달합니다.

Timer 클래스

ActionScript 3.0에서 시간 함수를 처리하는 좋은 방법은 지정된 시간 간격에 도달할 때마다 이벤트를 전달하는 Timer 클래스(flash.utils.Timer)를 사용하는 것입니다.

타이머를 시작하려면 먼저 Timer 클래스의 인스턴스를 생성하여 타이머 이벤트 생성 빈도 및 중단 시까지의 생성 횟수를 지정합니다.

예를 들어, 다음 코드는 매 초마다 이벤트를 전달하고 이 작업을 60초 동안 계속하는 Timer 인스턴스를 생성합니다.

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Timer 객체는 지정된 간격에 도달할 때마다 TimerEvent 객체를 전달합니다. TimerEvent 객체의 이벤트 유형은 `timer`(`TimerEvent.TIMER` 상수로 정의됨)입니다. TimerEvent 객체에는 표준 Event 객체와 동일한 속성이 들어 있습니다.

Timer 인스턴스가 고정된 간격 수로 설정된 경우에는 최종 간격에 도달할 때 `timerComplete` 이벤트(`TimerEvent.TIMER_COMPLETE` 상수로 정의됨)도 전달됩니다.

다음은 Timer 클래스의 작동을 보여주는 샘플 응용 프로그램입니다.

```
package
{
  import flash.display.Sprite;
  import flash.events.TimerEvent;
  import flash.utils.Timer;

  public class ShortTimer extends Sprite
  {
    public function ShortTimer()
    {
      // 5초 Timer를 새로 만듭니다.
      var minuteTimer:Timer = new Timer(1000, 5);

      // 간격 및 완료 이벤트에 대한 리스너를 지정합니다.
      minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
      minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE,
  onTimerComplete);

      // 타이머 티킹을 시작합니다.
      minuteTimer.start();
    }

    public function onTick(event:TimerEvent):void
    {
      // 현재까지의 틱 카운트를 표시합니다.
      // 이 이벤트의 대상은 Timer 인스턴스 자체입니다.
      trace("tick " + event.target.currentCount);
    }

    public function onTimerComplete(event:TimerEvent):void
    {
      trace("Time's Up!");
    }
  }
}
```

ShortTimer 클래스를 생성하면 5초 동안 1초에 한 번 작동하는 Timer 인스턴스가 생성됩니다. 그런 다음 타이머에 두 개의 리스너를 추가하여 하나는 각 틱(tick)을 수신하고 다른 하나는 `timerComplete` 이벤트를 수신하도록 합니다.

그러면 타이머 티킹이 시작되어 그 시점 이후부터 `onTick()` 메서드가 1초 간격으로 실행됩니다.

`onTick()` 메서드는 단순히 현재의 틱 카운트를 표시합니다. 5초가 지난 후에는 `onTimerComplete()` 메서드가 실행되어 시간이 종료되었음을 알립니다.

이 샘플을 실행하면 다음과 같은 행이 콘솔이나 추적 윈도우에 1초에 한 행씩 나타나는 것을 볼 수 있습니다.

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

flash.utils 패키지의 시간 함수

ActionScript 3.0에는 ActionScript 2.0에 사용된 시간 함수와 유사한 여러 함수가 포함되어 있습니다. 이러한 함수는 flash.utils 패키지의 패키지 레벨 함수로 제공되며 ActionScript 2.0에서와 동일한 방식으로 작동됩니다.

함수	설명
`clearInterval(id:uint):void`	지정된 `setInterval()` 호출을 취소합니다.
`clearTimeout(id:uint):void`	지정된 `setTimeout()` 호출을 취소합니다.
`getTimer():int`	Adobe Flash Player를 초기화한 이후 경과된 밀리초 수를 반환합니다.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	지정된 밀리초 단위 간격으로 함수를 실행합니다.
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	밀리초 단위로 지정된 지연 시간 후 지정된 함수를 실행합니다.

이러한 함수는 이전 버전과의 역호환성을 위해 ActionScript 3.0에 남아 있습니다. 하지만 새 ActionScript 3.0 응용 프로그램에는 이러한 함수를 사용하지 않는 것이 좋습니다. 일반적으로 응용 프로그램에 Timer 클래스를 사용하는 것이 보다 쉽고 효율적입니다.

예제: 간단한 아날로그 시계

간단한 아날로그 시계 예제를 통해 이 장에서 설명한 날짜와 시간의 두 개념에 대해 설명합니다.

- 현재 날짜 및 시간을 가져와서 시간, 분, 초 값으로 추출
- 타이머를 사용하여 응용 프로그램 속도 설정

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. SimpleClock 응용 프로그램 파일은 Samples/SimpleClock 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
SimpleClockApp.mxml 또는 SimpleClockApp.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/simpleclock/SimpleClock.as	기본 응용 프로그램 파일입니다.
com/example/programmingas3/simpleclock/AnalogClockFace.as	시간에 따라 둥근 시계 문자판과 시침, 분침, 초침을 그립니다.

SimpleClock 클래스 정의

시계 예제는 간단한 프로그램이지만, 아주 간단한 응용 프로그램이라도 나중에 쉽게 확장할 수 있도록 잘 구성하는 것이 좋습니다. 이 예제에서 SimpleClock 응용 프로그램은 SimpleClock 클래스를 사용하여 시작 및 시간 맞추기 작업을 처리한 다음 AnalogClockFace라는 다른 클래스를 사용하여 실제로 시간을 표시합니다.

다음은 SimpleClock 클래스를 정의하고 초기화하는 코드입니다. (Flash 버전에서 SimpleClock을 실행하면 Sprite 클래스가 확장됩니다.)

```
public class SimpleClock extends UIComponent
{
    /**
     * 시간 표시 구성 요소
     */
    private var face:AnalogClockFace;

    /**
     * 응용 프로그램의 심박 같은 기능을 하는 Timer
     */
    private var ticker:Timer;
```

이 클래스에는 다음과 같은 중요한 두 가지 속성이 들어 있습니다.

- face 속성(AnalogClockFace 클래스의 인스턴스임)
- ticker 속성(Timer 클래스의 인스턴스임)

SimpleClock 클래스는 기본 생성자를 사용합니다. initClock() 메서드는 실제 설정 작업을 처리하여 시계 문자판을 만들고 Timer 인스턴스 티킹을 시작합니다.

시계 문자판 만들기

SimpleClock 코드의 다음 행에서는 시간을 표시하는 데 사용하는 시계 문자판을 만듭니다.

```
/**
 * SimpleClock 인스턴스를 설정합니다 .
 */
public function initClock(faceSize:Number = 200)
{
    // 시계 문자판을 만들어서 표시 목록에 추가합니다 .
    face = new AnalogClockFace(Math.max(20, faceSize));
    face.init();
    addChild(face);

    // 초기 시계 문자판을 그립니다 .
    face.draw();
```

문자판의 크기는 initClock() 메서드로 전달될 수 있습니다. 전달된 faceSize 값이 없는 경우에는 기본 크기인 200픽셀이 사용됩니다.

그런 다음 응용 프로그램은 해당 문자판을 초기화한 후 DisplayObject 클래스에서 상속된 addChild() 메서드를 사용하여 표시 목록에 추가합니다. 그리고 나서 AnalogClockFace.draw() 메서드를 호출하여 시계 문자판을 현재 시간과 함께 한 번 표시합니다.

타이머 시작

시계 문자판을 만든 후에는 initClock() 메서드에서 타이머를 설정합니다.

```
    // 매초마다 한 번씩 이벤트를 발생시키는 Timer 를 만듭니다 .
    ticker = new Timer(1000);

    // Timer 이벤트를 처리할 onTick() 메서드를 지정합니다 .
    ticker.addEventListener(TimerEvent.TIMER, onTick);

    // 시계 티킹을 시작합니다 .
    ticker.start();
```

이 메서드는 먼저 1초(1000밀리초)에 한 번 이벤트를 전달하는 Timer 인스턴스를 인스턴스화합니다. 다른 `repeatCount` 매개 변수가 `Timer()` 생성자로 전달되지 않으므로 타이머는 무한정으로 계속 반복됩니다.

`timer` 이벤트를 수신하면 `SimpleClock.onTick()` 메서드가 1초에 한 번씩 실행됩니다.

```
public function onTick(event:TimerEvent):void
{
  // 시계 문자판을 업데이트합니다 .
  face.draw();
}
```

`AnalogClockFace.draw()` 메서드는 간단히 시계 문자판과 시계 바늘을 그립니다.

현재 시간 표시

AnalogClockFace 클래스 코드의 대부분은 시계 문자판의 표시 요소를 설정하는 것과 관련되어 있습니다. AnalogClockFace를 시작하면 둥근 외곽선을 그리고 각 시간 표시 부분에 숫자 텍스트 레이블을 놓은 다음 각각 시계의 시침, 분침, 초침에 해당하는 세 개의 Shape 객체를 생성합니다.

SimpleClock 응용 프로그램을 실행하면 다음과 같이 매 초마다 `AnalogClockFace.draw()` 메서드가 호출됩니다.

```
/**
 * 디스플레이가 그려질 때 부모 컨테이너에서 호출됩니다 .
 */
public override function draw():void
{
  // 인스턴스 변수에 현재 시간과 날짜를 저장합니다 .
  currentTime = new Date();
  showTime(currentTime);
}
```

이 메서드는 현재 시간을 변수로 저장하므로 시계 바늘을 그리는 도중에는 시간을 변경할 수 없습니다. 그런 다음 `showTime()` 메서드를 호출하여 다음과 같이 시계 바늘을 표시합니다.

```
/**
 * 주어진 날짜 / 시간을 옛날식 아날로그 시계 스타일로 표시합니다 .
 */
public function showTime(time:Date):void
{
  // 시간 값을 구합니다 .
  var seconds:uint = time.getSeconds();
  var minutes:uint = time.getMinutes();
  var hours:uint = time.getHours();

  // 6 을 곱해 각도를 구합니다 .
  this.secondHand.rotation = 180 + (seconds * 6);
  this.minuteHand.rotation = 180 + (minutes * 6);
```

```
    // 30을 곱해 기본 각도를 구한 다음 29.5도(59 * 0.5)까지
    // 더해 분을 계산합니다.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
  }
```

이 메서드는 먼저 현재 시간의 시, 분, 초 값을 추출합니다. 그런 다음 이 값을 사용하여 각 시계 바늘의 각도를 계산합니다. 초침은 60초에 한 번 회전하기 때문에 각 초마다 6도(360/60)씩 움직이게 됩니다. 분침은 매 분마다 동일한 각도씩 움직입니다.

시침도 매 분 업데이트되어 분침이 이동할 때마다 일정 정도 진행됩니다. 시침은 매 시간마다 30도(360/12)씩 움직이고 매 분마다 0.5도(30도/60분)씩 움직입니다.

제 6 장

문자열을 사용한 작업

6

String 클래스에는 텍스트 문자열을 사용하여 작업할 수 있도록 하는 메서드가 포함되어 있습니다. 문자열은 많은 객체를 사용하는 작업에 중요합니다. 이 장에서 설명된 메서드는 TextField, StaticText, XML, ContextMenu, FileReference와 같은 객체에 사용된 문자열을 사용하여 작업할 때 유용합니다.

문자열은 문자의 시퀀스입니다. ActionScript 3.0에서는 ASCII 및 유니코드 문자가 지원됩니다.

목차

문자열의 기초

문자열을 사용한 작업 소개

프로그래밍 용어에서 문자열은 텍스트 값, 즉 단일 값으로 묶인 일련의 문자, 숫자 또는 기타 문자를 뜻합니다. 예를 들어, 다음 코드 행은 데이터 유형이 String인 변수를 만들고 리터럴 문자열 값을 해당 변수에 지정합니다.

```
var albumName:String = "Three for the money";
```

이 예제에서 보듯이 ActionScript에서는 큰따옴표나 작은따옴표로 텍스트를 묶어서 문자열 값을 나타낼 수 있습니다. 다음은 몇 가지 문자열의 예입니다.

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

ActionScript에서 텍스트를 조작하는 경우 문자열 값으로 작업하는 것입니다. ActionScript String 클래스는 텍스트 값을 다룰 때 사용할 수 있는 데이터 유형입니다. 문자열 인스턴스는 다른 많은 ActionScript 클래스에서 속성, 메서드 매개 변수 등에 자주 사용됩니다.

문자열과 관련된 일반적인 작업

이 장에서 다루고 있는 문자열과 관련된 일반 작업은 다음과 같습니다.

- String 객체 만들기
- 캐리지 리턴, 탭 문자 및 키보드 이외 문자 등의 특수 문자 다루기
- 문자열 길이 측정
- 문자열에서 개별 문자 분리
- 문자열 연결
- 문자열 비교
- 문자열 일부분 찾기, 추출 및 바꾸기
- 문자열 대소문자 변경

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- ASCII: 컴퓨터 프로그램에서 텍스트 문자 및 기호를 나타내기 위한 시스템입니다. ASCII 시스템은 26자의 영어 알파벳과 제한된 추가 문자 집합을 지원합니다.
- 문자: 텍스트 데이터의 최소 단위입니다(단일 문자 또는 기호).

- 연결: 문자열 뒤에 다른 문자열을 추가하여 새로운 문자열 값을 만들 때와 같이 여러 문자열 값을 결합하는 것을 뜻합니다.
- 빈 문자열: 텍스트가 없거나 공백 또는 기타 문자가 포함된 문자열이며 ""로 작성됩니다. 빈 문자열은 null 값을 가진 String 변수와 다릅니다. null 값의 String 변수는 변수에 지정된 문자열 인스턴스가 없는 변수지만, 빈 문자열은 값에 문자가 포함되지 않은 인스턴스를 가집니다.
- 문자열: 텍스트 값입니다(일련의 문자).
- 문자열 리터럴(또는 "리터럴 문자열"): 코드에서 명시적으로 작성된 문자열로, 텍스트값을 큰따옴표 또는 작은따옴표로 묶어 표기합니다.
- 하위 문자열: 다른 문자열이 일부가 되는 문자열입니다.
- 유니코드: 컴퓨터 프로그램에서 텍스트 문자 및 기호를 나타내기 위한 표준 시스템입니다. 유니코드 시스템에서는 모든 쓰기 시스템에서 어느 문자든지 사용할 수 있습니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플은 주로 텍스트 조작을 다루므로, 예제를 테스트하려면 텍스트 필드 인스턴스의 값을 스테이지에 기록하거나 `trace()` 함수를 사용하여 [출력] 패널에 값을 출력함으로써 해당 예제에 사용되는 변수 값을 표시해야 합니다. 이러한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

문자열 만들기

String 클래스는 ActionScript 3.0에서 문자열(텍스트) 데이터를 나타내는 데 사용됩니다. ActionScript 문자열에서는 ASCII 및 유니코드 문자가 모두 지원됩니다. 문자열을 만드는 가장 간단한 방법은 문자열 리터럴을 사용하는 것입니다. 문자열 리터럴을 선언하려면 곧은 큰따옴표(")나 작은따옴표(') 문자를 사용합니다. 예를 들어, 다음 두 문자열은 동일합니다.

```
var str1:String = "hello";
var str2:String = 'hello';
```

다음과 같이 `new` 연산자를 사용하여 문자열을 선언할 수도 있습니다.

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

다음 두 문자열은 동일합니다.

```
var str1:String = "hello";
var str2:String = new String("hello");
```

작은따옴표(') 구분 기호로 정의된 문자열 리터럴에 작은따옴표(')를 사용하려면 이스케이프 문자인 백슬래시(\)를 사용합니다. 또한 큰따옴표(") 구분 기호로 정의된 문자열 리터럴에 큰따옴표(")를 사용할 때도 이스케이프 문자인 백슬래시(\)를 사용합니다. 다음 두 문자열은 동일합니다.

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

다음과 같이 문자열 리터럴에 있는 작은따옴표나 큰따옴표를 기준으로 작은따옴표나 큰따옴표를 선택하여 사용할 수 있습니다.

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

ActionScript에서는 곧은 작은따옴표(')와 왼쪽이나 오른쪽 작은따옴표(‘ 또는 ’)가 구분됩니다. 큰따옴표의 경우에도 마찬가지입니다. 문자열 리터럴을 구분하려면 곧은 따옴표를 사용합니다. 다른 소스의 텍스트를 ActionScript로 붙여 넣을 경우 정확한 문자를 사용해야 합니다.

다음 표와 같이 이스케이프 문자인 백슬래시(\)를 사용하여 문자열 리터럴에 다른 문자를 정의할 수 있습니다.

이스케이프 시퀀스	문자
\b	백스페이스
\f	용지 공급
\n	개행
\r	캐리지 리턴
\t	탭
\u*nnnn*	16진수 *nnnn*으로 지정된 문자 코드(예: \u263a)가 있는 유니코드 문자는 스마일리 문자입니다.
\x*nn*	16진수 *nn*으로 지정된 문자 코드가 있는 ASCII 문자
\'	작은따옴표
\"	큰따옴표
\\	단일 백슬래시 문자

length 속성

모든 문자열에는 `length` 속성이 있습니다. 이 속성은 문자열에 있는 문자의 수와 같습니다.

```
var str:String = "Adobe";
trace(str.length);              // 출력 : 5
```

빈 문자열 및 null 문자열의 길이는 다음 예제와 같이 0입니다.

```
var str1:String = new String();
trace(str1.length);             // 출력 : 0

str2:String = '';
trace(str2.length);             // 출력 : 0
```

문자열 내의 문자 작업

문자열의 모든 문자에는 인덱스 위치(정수)가 있습니다. 첫 번째 문자의 인덱스 위치는 0입니다. 예를 들어, 다음 문자열에서 문자 `y`는 위치 0에 있고 문자 `w`는 위치 5에 있습니다.

```
"yellow"
```

다음 예제와 같이 `charAt()` 메서드 및 `charCodeAt()` 메서드를 사용하여 문자열 내의 다양한 위치에서 개별 문자를 검사할 수 있습니다.

```
var str:String = "hello world!";
for (var:i = 0; i < str.length; i++)
{
  trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

이 코드를 실행하면 다음과 같이 출력됩니다.

```
h - 104
e - 101
l - 108
l - 108
o - 111
  - 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

다음 예제와 같이 문자 코드를 사용하여 `fromCharCode()` 메서드를 통해 문자열을 정의할 수도 있습니다.

```
var myStr:String =
  String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
    // myStr을 "hello world!"로 설정합니다.
```

문자열 비교

<, <=, !=, ==, => 및 > 연산자를 사용하여 문자열을 비교할 수 있습니다. 이러한 연산자는 다음 예제와 같이 `if` 및 `while`과 같은 조건문과 함께 사용할 수 있습니다.

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
  trace("A < a, B < b, C < c, ...");
}
```

문자열에서 이러한 연산자를 사용할 경우 ActionScript에서는 다음과 같이 문자열에 있는 각 문자의 문자 코드 값을 고려하여 왼쪽부터 오른쪽 순으로 문자를 비교합니다.

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

다음 예제와 같이 문자열을 서로 비교하고 문자열을 다른 유형의 객체와 비교하려면 == 및 != 연산자를 사용합니다.

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

다른 객체의 문자열 표현 가져오기

모든 종류의 객체에 대한 문자열 표현을 가져올 수 있습니다. 모든 객체에는 이 용도로 사용되는 `toString()` 메서드가 있습니다.

```
var n:Number = 99.47;
var str:String = n.toString();
  // str == "99.47"
```

String 객체와 문자열이 아닌 객체를 결합하기 위해 + 연결 연산자를 사용할 때 `toString()` 메서드를 사용하지 않아도 됩니다. 연결에 대한 자세한 내용은 다음 단원을 참조하십시오.

`String()` 전역 함수는 지정된 객체에 대해 `toString()` 메서드를 호출하여 반환된 값과 동일한 값을 반환합니다.

문자열 연결

문자열 연결은 두 개의 문자열을 가지고 순차적으로 하나로 결합하는 것을 의미합니다. 예를 들어, + 연산자를 사용하여 두 문자열을 연결할 수 있습니다.

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

+= 연산자를 사용해도 다음 예제와 같이 동일한 결과를 생성할 수 있습니다.

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

또한 String 클래스에는 `concat()` 메서드가 포함되며, 다음과 같이 사용될 수 있습니다.

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

String 객체와 문자열이 *아닌* 객체에 + 연산자(또는 += 연산자)를 사용할 경우 ActionScript에서는 다음 예제와 같이 표현식을 평가하기 위해 문자열이 아닌 객체를 String 객체로 자동 변환합니다.

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

하지만 다음 예제와 같이 그룹 괄호를 사용하여 + 연산자에 대한 컨텍스트를 제공할 수 있습니다.

```
trace("Total: $" + 4.55 + 1.45); // 출력 : Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // 출력 : Total: $6
```

문자열의 패턴 및 하위 문자열 찾기

하위 문자열은 문자열 내의 연속 문자입니다. 예를 들어, 문자열 `"abc"`에는 `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`와 같은 하위 문자열이 있습니다. ActionScript 메서드를 사용하여 문자열의 하위 문자열을 찾을 수 있습니다.

패턴은 ActionScript에서 문자열이나 일반 표현식으로 정의됩니다. 예를 들어, 다음 일반 표현식은 글자 A, B, C 다음에 숫자가 오는 특정 패턴을 정의합니다(슬래시는 일반 표현식 구분 기호임).

```
/ABC\d/
```

ActionScript에는 문자열의 패턴을 찾고 찾은 항목을 대체 하위 문자열로 바꾸는 메서드가 포함되어 있습니다. 이러한 메서드에 대해서는 다음 단원에서 설명합니다.

일반 표현식은 복잡한 패턴을 정의할 수 있습니다. 자세한 내용은 269페이지의 제9장, “일반 표현식 사용”을 참조하십시오.

문자 위치로 하위 문자열 찾기

substr() 및 substring() 메서드는 서로 비슷합니다. 두 메서드 모두 문자열의 하위 문자열을 반환합니다. 또한 두 개의 매개 변수를 사용합니다. 두 메서드에서 첫 번째 매개 변수는 지정된 문자열에서 시작 문자의 위치입니다. 그러나 substr() 메서드에서 두 번째 매개 변수는 반환할 하위 문자열의 *길이*인 반면 substring() 메서드에서 두 번째 매개 변수는 반환된 문자열에 포함되지 않는 하위 문자열의 *끝*에 있는 문자의 위치입니다. 다음 예제에서는 두 메서드 간의 차이점을 보여 줍니다.

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // 출력 : Paris, Texas!!!
trace(str.substring(11,15)); // 출력 : Pari
```

slice() 메서드는 substring() 메서드와 비슷하게 작동합니다. 두 개의 음수가 아닌 정수를 매개 변수로 지정하면 두 메서드는 똑같이 작동합니다. 하지만 slice() 메서드는 매개 변수로 음의 정수를 사용할 수 있으며, 이 경우 문자 위치는 다음 예제에서와 같이 문자열 끝에서 가지고 옵니다.

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // 출력 : Pari
trace(str.slice(-3,-1)); // 출력 : !!
trace(str.slice(-3,26)); // 출력 : !!!
trace(str.slice(-3,str.length)); // 출력 : !!!
trace(str.slice(-8,-3)); // 출력 : Texas
```

slice() 메서드의 매개 변수로 음수가 아닌 정수와 음의 정수를 결합하여 사용할 수 있습니다.

일치하는 하위 문자열의 문자 위치 찾기

indexOf() 및 lastIndexOf() 메서드를 사용하여 다음 예제와 같이 한 문자열 내에서 일치하는 하위 문자열을 찾을 수 있습니다.

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // 출력 : 10
```

indexOf() 메서드는 대/소문자를 구분합니다.

두 번째 매개 변수를 지정하여 다음과 같이 검색을 시작할 문자열의 인덱스 위치를 나타낼 수 있습니다.

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // 출력 : 21
```

lastIndexOf() 메서드는 문자열에서 마지막 하위 문자열을 찾습니다.

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // 출력 : 30
```

lastIndexOf() 메서드로 두 번째 매개 변수를 포함하면 문자열의 해당 인덱스 위치부터 역방향(오른쪽에서 왼쪽)으로 검색이 실행됩니다.

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // 출력 : 21
```

구분 기호로 분리된 하위 문자열 배열 만들기

split() 메서드를 사용하여 하위 문자열의 배열을 만들 수 있습니다. 하위 문자열은 구분 기호를 기준으로 분리됩니다. 예를 들어, 쉼표로 구분되거나 탭으로 구분된 문자열을 여러 문자열로 분리할 수 있습니다.

다음 예제에서는 구분 기호로 앰퍼샌드(&) 문자를 사용하여 배열을 하위 문자열로 분리하는 방법을 보여 줍니다.

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/
  65";
var params:Array = queryStr.split("&", 2); // params ==
  ["first=joe","last=cheng"]
```

split() 메서드의 두 번째 매개 변수는 선택 사항이며 반환되는 배열의 최대 크기를 정의합니다.

구분 기호 문자로 일반 표현식을 사용할 수도 있습니다.

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

자세한 내용은 269페이지의 제9장, “일반 표현식 사용” 및 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

문자열의 패턴 찾기 및 하위 문자열 바꾸기

String 클래스에는 문자열의 패턴을 사용하여 작업하는 다음과 같은 메서드가 포함됩니다.

- `match()` 및 `search()` 메서드를 사용하여 패턴과 일치하는 하위 문자열을 찾습니다.
- `replace()` 메서드를 사용하여 패턴과 일치하는 하위 문자열을 찾고 지정한 하위 문자열로 바꿉니다.

이러한 메서드에 대해서는 다음 단원에서 설명합니다.

문자열이나 일반 표현식을 사용하여 이러한 메서드에 사용할 패턴을 정의할 수 있습니다. 일반 표현식에 대한 자세한 내용은 269페이지의 제9장, “일반 표현식 사용”을 참조하십시오.

일치하는 하위 문자열 찾기

`search()` 메서드는 이 예제와 같이 지정된 패턴과 일치하는 첫 번째 하위 문자열의 인덱스 위치를 반환합니다.

```
var str:String = "The more the merrier.";
// (이 검색은 대/소문자를 구분합니다.)
trace(str.search("the")); // 출력: 9
```

일반 표현식을 사용하여 이 예제와 같이 일치하는 패턴을 정의할 수도 있습니다.

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

문자열의 첫 번째 문자는 인덱스 위치 0이므로 `trace()` 메서드의 출력은 0이며, `i` 플래그가 일반 표현식에 설정되어 있으므로, 검색 시 대/소문자를 구분하지 않습니다.

`search()` 메서드는 `g`(전역) 플래그가 일반 표현식에 설정되어 있어도 일치하는 한 항목만 찾으며 그 시작 인덱스 위치를 반환합니다.

다음 예제에서는 큰따옴표로 묶인 문자열과 일치하는 보다 복잡한 일반 표현식을 보여 줍니다.

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // 출력: 4


str = "The \"more the merrier.";
trace(str.search(pattern)); // 출력: -1
// (닫는 큰따옴표가 없어 일치하는 항목이 없습니다.)
```

`match()` 메서드는 이와 유사하게 작동하며, 일치하는 하위 문자열을 검색합니다. 하지만 다음 예제와 같이 일반 표현식 패턴에서 전역 플래그를 사용할 경우 `match()` 메서드는 일치하는 하위 문자열의 배열을 반환합니다.

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

results 배열은 다음과 같이 설정됩니다.

```
["bob@example.com","omar@example.org"]
```

일반 표현식에 대한 자세한 내용은 269페이지의 제9장, “일반 표현식 사용”을 참조하십시오.

일치하는 하위 문자열 바꾸기

replace() 메서드를 사용하여 다음 예제와 같이 문자열에서 지정된 패턴을 검색하고 지정된 대체 문자열로 일치 항목을 바꿀 수 있습니다.

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
  //sche sells seaschells by the seaschore.
```

이 예제에서는 i(ignoreCase) 플래그가 일반 표현식에 설정되어 있으므로 일치 문자열이 대/소문자를 구분하지 않으며, g(global) 플래그가 설정되어 있으므로 여러 일치 항목이 모두 바뀝니다. 자세한 내용은 269페이지의 제9장, “일반 표현식 사용”을 참조하십시오.

대체 문자열에서 다음 $ 대체 코드를 사용할 수 있습니다. 다음 표에 표시된 대체 텍스트는 $ 대체 코드 위치로 삽입됩니다.

$ 코드	대체 텍스트
$$	$
$&	일치하는 하위 문자열
$`	일치하는 하위 문자열 앞에 오는 문자열의 일부. 이 코드에는 곧은 작은따옴표(')나 왼쪽으로 굽은 작은따옴표(‘)가 아닌 왼쪽으로 기울고 곧은 작은따옴표 문자(`)가 사용됩니다.
$'	일치하는 하위 문자열 다음에 오는 문자열의 일부. 이 코드에는 곧은 작은따옴표(')가 사용됩니다.
$*n*	괄호로 둘러싼 일치 그룹 중 *n*번째로 캡처된 항목입니다. 여기에서 *n*은 1에서 9 사이의 한 자리 숫자이고 $*n* 다음에는 10진수 숫자가 오지 않습니다.
$*nn*	괄호로 둘러싼 일치 그룹 중 *nn*번째로 캡처된 항목입니다. 여기에서 *nn*은 01에서 99 사이의 두 자리 10진수 숫자입니다. *nn*번째 캡처가 정의되지 않은 경우 대체 텍스트는 빈 문자열입니다.

예를 들어, 다음 예제에서는 일치 그룹 중 첫 번째 및 두 번째 캡처 항목을 나타내는 $2 및 $1 대체 코드를 사용하는 방법을 보여 줍니다.

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

함수를 replace() 메서드의 두 번째 매개 변수로 사용할 수도 있습니다. 이 경우 일치하는 텍스트는 반환된 함수 값으로 바뀝니다.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,
                   capturedMatch1:String,
                   index:int,
                   str:String):String
{
  var usd:String = capturedMatch1;
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.853690;
  var euro:Number = usd * exchangeRate;
  const euroSymbol:String = String.fromCharCode(8364);
  return euro.toFixed(2) + " " + euroSymbol;
}
```

함수를 replace() 메서드의 두 번째 매개 변수로 사용할 때 다음 인수가 함수로 전달됩니다.

- 문자열의 일치 부분
- 괄호로 둘러싼 일치 그룹 중 캡처할 항목. 이렇게 전달되는 인수의 수는 괄호로 둘러싼 일치 항목의 수에 따라 달라집니다. 괄호로 둘러싼 일치 항목의 수를 확인하려면 함수 코드 내에서 arguments.length - 3을 확인합니다.
- 문자열에서 검색을 시작할 인덱스 위치
- 전체 문자열

대/소문자 간 문자열 변환

다음 예제와 같이 toLowerCase() 메서드 및 toUpperCase() 메서드는 문자열에 있는 알파벳 문자를 각각 소문자와 대문자로 변환합니다.

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

이 메서드를 실행하더라도 소스 문자열은 변경되지 않습니다. 소스 문자열을 변환하려면 다음 코드를 사용합니다.

```
str = str.toUpperCase();
```

이러한 메서드는 단순히 a-z 및 A-Z 문자뿐 아니라 확장 문자에도 적용됩니다.

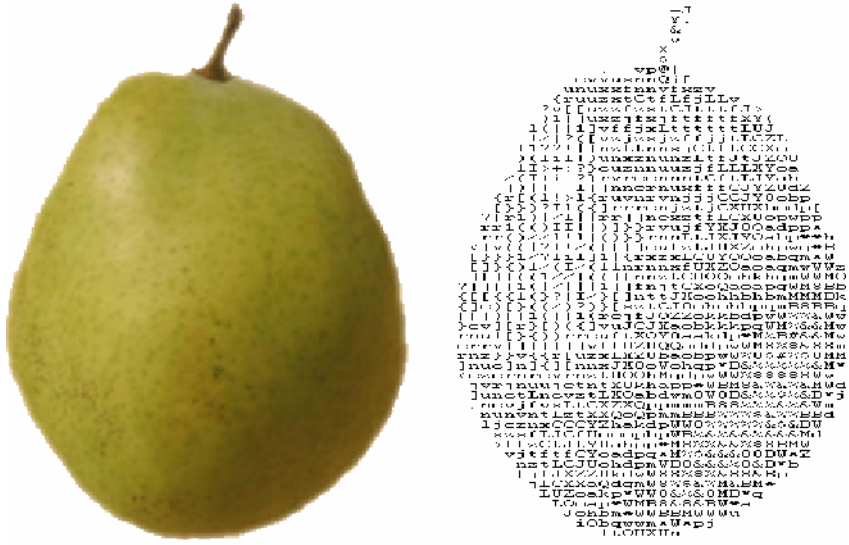
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

예제: ASCII Art

ASCII Art 예제에서는 ActionScript 3.0에서 String 클래스를 사용한 작업의 다양한 기능을 보여 줍니다. 다음은 그 기능 중 일부입니다.

- String 클래스의 `split()` 메서드는 문자 구분 문자열에서 값을 추출(탭 구분 텍스트 파일의 이미지 정보)하는 데 사용됩니다.
- `split()`, 연결 및 `substring()`, `substr()`을 사용한 문자열 일부 추출 등을 비롯한 여러 문자열 조작 기술은 이미지 제목에서 각 단어의 첫 번째 글자를 대문자화하는 데 사용됩니다.
- `getCharAt()` 메서드는 문자열에서 단일 문자를 가져오는 데 사용됩니다(회색 음영 비트맵 값에 해당하는 ASCII 문자를 결정하기 위해).
- 문자열 연결은 한 번에 하나의 문자씩 이미지의 ASCII Art 표현을 작성하는 데 사용됩니다.

*ASCII Art*는 Courier New 문자와 같은 고정폭 글꼴 문자로 구성된 격자를 사용하여 이미지를 표시하는 것으로, 이미지를 텍스트로 표현한 것을 나타냅니다. 다음 이미지는 응용 프로그램에서 생성된 ASCII Art의 예입니다.



오른쪽 이미지는 그래픽의 ASCII Art 버전입니다.

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. ASCIIArt 응용 프로그램 파일은 Samples/AsciiArt 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
AsciiArtApp.mxml 또는 AsciiArtApp.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	텍스트 파일의 이미지 메타데이터 추출, 이미지 로딩, 이미지-텍스트 변환 프로세스 관리 등을 비롯한 응용 프로그램의 주요 기능을 제공하는 클래스입니다.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	이미지 데이터를 문자열 버전으로 변환하기 위한 `parseBitmapData()` 메서드를 제공하는 클래스입니다.
com/example/programmingas3/asciiArt/Image.as	로드된 비트맵 이미지를 나타내는 클래스입니다.
com/example/programmingas3/asciiArt/ImageInfo.as	제목, 이미지 파일 URL 등과 같은 ASCII Art 이미지에 대한 메타데이터를 나타내는 클래스입니다.
image/	응용 프로그램에서 사용된 이미지가 포함되어 있는 폴더입니다.
txt/ImageData.txt	응용 프로그램에 로드되는 이미지에 대한 정보가 포함되어 있는, 탭으로 구분된 텍스트 파일입니다.

탭 구분 값 추출

이 예제에서는 응용 프로그램 자체와 응용 프로그램 데이터를 별도로 저장하는 일반적인 규칙이 사용되어, 데이터가 변경된 경우에도(예: 다른 이미지가 추가되거나 이미지 제목이 변경된 경우) SWF 파일을 다시 만들 필요가 없습니다. 이 경우 이미지 제목, 실제 이미지 파일의 URL, 이미지를 조작하는 데 사용된 일부 값 등을 비롯한 이미지 메타데이터는 텍스트 파일에 저장됩니다(프로젝트의 txt/ImageData.txt 파일). 이 텍스트 파일의 내용은 다음과 같습니다.

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

이 파일은 특정 탭 구분 포맷을 사용합니다. 첫 번째 줄(행)은 머리글 행입니다. 나머지 행에는 로드할 각 비트맵에 대한 다음 데이터가 포함됩니다.

- 비트맵의 파일 이름
- 비트맵의 표시 이름
- 비트맵의 흰색 임계값 및 검정 임계값. 16진수 값으로 이 값 위 및 아래의 픽셀은 완전한 흰색이나 완전한 검정으로 간주됩니다.

응용 프로그램이 시작되면 즉시 AsciiArtBuilder 클래스에서는 AsciiArtBuilder 클래스의 `parseImageInfo()` 메서드의 다음 코드를 사용하여, 표시할 이미지의 "스택"을 만들기 위해 텍스트 파일의 내용을 로드하고 파싱합니다.

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
  var imageInfoRaw:String = lines[i];
  ...
  if (imageInfoRaw.length > 0)
  {
    // 새 이미지 정보 레코드를 만들어서 이미지 정보 배열에 추가합니다.
    var imageInfo:ImageInfo = new ImageInfo();

    // 현재 행을 탭(\t 문자)으로 구분된 여러 값으로 분할하고
    // 개별 속성을 추출합니다.
    var imageProperties:Array = imageInfoRaw.split("\t");
    imageInfo.fileName = imageProperties[0];
    imageInfo.title = normalizeTitle(imageProperties[1]);
    imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
    imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
    result.push(imageInfo);
  }
}
```

텍스트 파일의 전체 내용은 단일 문자열 인스턴스, `_imageInfoLoader.data` 속성에 포함됩니다. 개행 문자("\n")를 매개 변수로 한 `split()` 메서드를 사용하여, 문자열 인스턴스를 각 요소가 텍스트 파일의 개별 행인 배열(`lines`)로 분할합니다. 그런 다음, 코드에서 루프를 사용하여 각 행에 대한 작업을 실행합니다(첫 번째 행은 실제 내용이 아닌 머리글만 포함되므로 제외). 루프 내에서 `split()` 메서드를 한 번 더 사용하여 한 행의 내용을 하나의 값 세트(`imageProperties`라는 Array 객체)로 분할합니다. 이때 `split()` 메서드와 함께 사용되는 매개 변수는 각 행의 값이 탭 문자로 구분되어 있으므로 탭("\t") 문자입니다.

String 메서드를 사용하여 이미지 제목 정규화

이 응용 프로그램의 디자인 결정 사항 중 하나는 모든 이미지 제목을 각 단어의 첫 번째 글자를 대문자로 표시하는 표준 포맷을 사용하여 표시하는 것입니다(영어 제목에서 일반적으로 대문자로 쓰이지 않는 일부 단어 제외). 텍스트 파일에 적절하게 포맷된 제목이 포함되었다고 가정하는 것이 아니라, 이 응용 프로그램은 텍스트 파일에서 제목을 추출할 때 직접 제목을 포맷합니다.

이전 코드 샘플에서 개별 이미지 메타데이터 값 추출의 일부로 다음 코드 행이 사용됩니다.

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

이 코드에서 텍스트 파일의 이미지 제목은 ImageInfo 객체에 저장되기 전에 `normalizeTitle()` 메서드를 통해 전달됩니다.

```
private function normalizeTitle(title:String):String
{
  var words:Array = title.split(" ");
  var len:uint = words.length;
  for (var i:uint; i < len; i++)
  {
    words[i] = capitalizeFirstLetter(words[i]);
  }

  return words.join(" ");
}
```

이 메서드는 `split()` 메서드를 사용하여 제목을 개별 단어(공백 문자로 구분)로 분리한 뒤 `capitalizeFirstLetter()` 메서드를 통해 각 단어를 전달하고 Array 클래스의 `join()` 메서드를 사용하여 단어를 단일 문자열로 다시 결합합니다.

이름에서 알 수 있듯이 `capitalizeFirstLetter()` 메서드는 각 단어의 첫 번째 글자를 대문자로 바꾸는 작업을 합니다.

```
  /**
   * 영어에서 일반적으로 소문자로 두는 단어를 제외하고
   * 단어의 첫 번째 글자를 대문자로 바꿉니다 .
   */
  private function capitalizeFirstLetter(word:String):String
  {
    switch (word)
    {
      case "and":
      case "the":
      case "in":
      case "an":
      case "or":
      case "at":
      case "of":
      case "a":
        // 이 단어에 대해 아무 작업도 수행하지 않습니다 .
```

```
            break;
        default:
            // 다른 단어의 경우 첫 문자를 대문자로 바꿉니다 .
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

영어에서는 제목에 "and," "the," "in," "an," "or," "at," "of," "a"와 같은 단어가 있을 경우 해당 단어의 첫 자는 대문자로 쓰지 *않습니다*(간소화된 규칙 버전). 이 논리를 실행하기 위해 코드에서는 먼저 `switch` 문을 사용하여 단어가 대문자로 쓰면 안 되는 단어인지 확인합니다. 대문자로 쓰면 안 되는 단어 중 하나일 경우 코드는 `switch` 문을 빠져 나옵니다. 반면 대문자로 써야 할 단어일 경우 다음과 같이 여러 단계를 수행합니다.

1. 단어의 첫 번째 글자가 `substr(0, 1)`을 사용하여 추출됩니다. 이 메서드는 인덱스 0에 있는 문자(첫 번째 매개 변수 `0`으로 지정된 대로 문자열의 첫 번째 글자)로 시작되는 하위 문자열을 추출합니다. 하위 문자열은 두 번째 매개 변수 `1`로 지정된 길이 대로 한 문자가 됩니다.
2. 이 문자를 `toUpperCase()` 메서드를 사용하여 대문자화합니다.
3. 원래 단어의 나머지 문자는 `substring(1)`을 사용하여 추출되며, 이 메서드는 인덱스 1에서 시작하는 하위 문자열(두 번째 글자)을 문자열의 끝까지(`substring()` 메서드의 두 번째 매개 변수를 생략하여 지정) 추출합니다.
4. 최종 단어는 문자열 연결(`firstLetter + otherLetters`)을 사용하여 대문자로 바뀐 첫 번째 글자와 나머지 글자를 결합하여 만들어집니다.

ASCII Art 텍스트 생성

BitmapToAsciiConverter 클래스는 비트맵 이미지를 ASCII 텍스트 표현으로 변환하는 기능을 제공합니다. 이 프로세스는 `parseBitmapData()` 메서드에서 수행되며 아래에 프로세스 일부가 나와 있습니다.

```
var result:String = "";

// 위에서 아래로 모든 픽셀 행에 대해 반복합니다 .
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // 각 행 내에서 왼쪽에서 오른쪽으로 모든 픽셀 행에 대해 반복합니다 .
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // 0-255 범위의 회색 값을 사용 가능한 문자
```

```
        // 세트의 회색 음영 수인 0-64 범위의 값으로 변환합니다 .
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
      }
      result += "\n";
    }
    return result;
```

이 코드는 비트맵 이미지의 ASCII Art 버전을 작성하는 데 사용될 `result`라는 문자열 인스턴스를 먼저 정의합니다. 그런 다음 소스 비트맵 이미지의 개별 픽셀에 대해 작업을 반복합니다. 즉, 여러 색상 조작 기술을 사용하여(지면 관계상 자세한 내용은 생략) 개별 픽셀의 빨강, 녹색, 파랑 색상 값을 단일 회색 음영 값(0-255)으로 변환합니다. 그런 다음 위의 코드와 같이 이 값을 4로 나누어 0-63 사이의 값으로 변환한 후, 변수 `index`에 저장합니다. 이 응용 프로그램에서 사용되는 사용 가능한 ASCII 문자 "팔레트"에 64개의 값이 포함되므로 0-63 사이의 값이 사용됩니다. 문자의 팔레트는 BitmapToAsciiConverter 클래스에서 문자열 인스턴스로 정의됩니다.

```
// 문자는 문자열에서의 위치 ( 인덱스 ) 가 상대 색상 값 (0 = 검정 ) 에 해당하도록
// 가장 어두운 문자에서 가장 밝은 문자의 순서로 표시됩니다 .
private static const palette:String =
  "@#$%&8BMW*mwqpdbkhaoQOOZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

`index` 변수가 비트맵 이미지의 현재 픽셀에 해당하는 팔레트의 ASCII 문자를 정의하므로, 문자는 `charAt()` 메서드를 사용하여 `palette` String에서 가져옵니다. 그런 다음 이 문자를 연결 대입 연산자(+=)를 사용하여 `result` 문자열 인스턴스에 추가합니다. 또한 각 픽셀 행의 끝에서, 개행 문자를 `result` String의 끝에 연결하여 새 문자 "픽셀" 행을 줄을 바꿔 표시하도록 합니다.

제 7 장

배열을 사용한 작업

7

배열을 통해 여러 값을 하나의 데이터 구조에 저장할 수 있습니다. 정해진 순서의 정수 인덱스를 사용하여 값을 저장하는 간단한 인덱스 배열이나 임의의 키를 사용하여 값을 저장하는 복잡한 연관 배열을 사용할 수 있습니다. 배열은 자체가 배열인 요소를 포함하는 다차원으로 구성될 수도 있습니다. 이 장에서는 다양한 유형의 배열을 만들고 조작하는 방법에 대해 설명합니다.

목차

배열의 기초

배열을 사용한 작업 소개

프로그래밍 작업 시에는 단일 객체보다는 항목 집합을 사용하여 작업해야 하는 경우가 많습니다. 예를 들어, 뮤직 플레이어 응용 프로그램을 개발할 때 재생할 노래 목록을 포함하고자 할 수 있습니다. 이 경우 노래 목록에 있는 각 노래에 대해 별도의 변수를 만들기 보다는 모든 노래 객체를 한 그룹으로 묶어 작업하는 것이 좋습니다.

배열이란 노래 목록과 같이 항목 집합의 컨테이너 역할을 하는 프로그래밍 요소를 의미합니다. 대체적으로 배열에 포함된 모든 항목은 동일한 클래스의 인스턴스가 되지만, ActionScript에서는 반드시 필요한 것은 아닙니다. 배열의 개별 항목은 배열 *요소*라고도 합니다. 다시 말해 배열을 변수 보관용 파일 서랍으로 생각할 수도 있습니다. 파일 서랍에 폴더를 넣듯이, 변수를 요소로서 배열에 추가할 수 있습니다. 서랍에 여러 파일이 채워지면 배열을 단일 변수로 사용하여 작업하거나(예: 서랍 전체를 다른 위치로 전달), 변수를 한 그룹으로 사용하여 작업하거나(예: 폴더를 한 페이지씩 넘기면서 정보 찾기), 각각의 변수에 개별적으로 액세스(예: 서랍을 열고 단일 폴더 선택)할 수 있습니다.

예를 들어, 여러 곡을 선택하여 재생 목록에 추가할 수 있는 뮤직 플레이어 응용 프로그램을 개발한다고 가정해 보겠습니다. ActionScript 코드에서 `addSongsToPlaylist()`라는 메서드를 사용할 수 있는데, 이 경우 단일 배열을 매개 변수로 받게 됩니다. 재생 목록에 추가할 곡 수가 몇 곡되지 않거나 아주 많거나 혹은 단 1곡이더라도, 이와는 관계없이 `addSongsToPlaylist()` 메서드를 한 번만 호출하여 Song 객체가 들어 있는 배열을 전달하게 됩니다. `addSongsToPlaylist()` 메서드에는 배열의 요소(노래)를 하나씩 확인하면서 재생 목록에 실제로 추가하는 루프를 포함시킬 수 있습니다.

가장 널리 사용되는 ActionScript 배열 유형은 번호가 지정된 슬롯(*인덱스*)에 각 항목을 저장하는 *인덱스 배열*입니다. 인덱스 배열에서는 번호를 주소처럼 사용하여 항목에 액세스합니다. 인덱스 배열을 나타내기 위해 Array 클래스가 사용됩니다. 인덱스 배열을 사용하면 프로그래밍에 필요한 대부분의 작업을 적절히 처리할 수 있습니다. 인덱스 배열을 특수한 형태로 활용한 것이 다차원 배열로, 인덱스 배열을 요소로 갖는 인덱스 배열을 나타냅니다. 즉 인덱스 배열 요소에는 다른 요소가 포함되게 됩니다. 또 다른 배열 유형으로 *연관 배열*이 있는데, 여기에서는 숫자 인덱스가 아닌 문자열 *키*를 사용하여 개별 요소를 식별합니다. 마지막으로 ActionScript 3.0에는 고급 사용자를 위한 Dictionary 클래스가 포함되어 있습니다. Dictionary 클래스는 문자 그대로 *사전*을 의미하며, 객체 유형을 키로 사용하여 요소를 구분할 수 있습니다.

일반적인 배열 작업

이 장에서 설명할 일반적인 배열 작업은 다음과 같습니다.

- 인덱스 배열 만들기
- 배열 요소 추가 및 제거
- 배열 요소 정렬
- 배열의 일부 추출
- 연관 배열 및 사전을 사용한 작업
- 다차원 배열을 사용한 작업
- 배열 요소 복사
- 배열의 하위 클래스 만들기

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 배열: 여러 객체를 함께 그룹화할 수 있도록 컨테이너 역할을 하는 객체입니다.
- 연관 배열: 문자열 키를 사용하여 개별 요소를 식별하는 배열입니다.
- 사전: 항목이 객체 쌍(키와 값)으로 구성된 배열입니다. 숫자 인덱스 대신 키를 사용하여 각 요소를 식별합니다.
- 요소: 배열 내의 각 항목을 나타냅니다.
- 인덱스: 인덱스 배열에 있는 각 항목을 식별하는 "주소"로 숫자 값을 가집니다.
- 인덱스 배열: 번호가 지정된 요소에 각 요소를 저장하는 표준 유형의 배열이며, 번호(인덱스)로 개별 요소를 식별합니다.
- 키: 연관 배열 또는 사전의 각 요소를 식별하는 문자열 또는 객체입니다.
- 다차원 배열: 단일 값이 아닌 배열로 구성된 항목이 들어 있는 배열을 나타냅니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 기본적으로 이 장의 모든 코드 샘플에는 해당하는 `trace()` 함수 호출이 포함되어 있습니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   [출력] 패널에서 `trace()` 함수의 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 모든 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

인덱스 배열

인덱스 배열은 일련의 하나 이상 값을 부호 없는 정수 값을 사용하여 각 값에 액세스할 수 있도록 구성하여 저장합니다. 첫 번째 인덱스는 항상 숫자 0이며 배열에 요소를 추가할 때마다 인덱스가 1씩 증가합니다. 다음 코드와 같이 Array 클래스 생성자를 호출하거나 배열 리터럴을 사용하여 배열을 초기화함으로써 인덱스 배열을 만들 수 있습니다.

```
// Array 생성자를 사용합니다 .
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // 출력 : one,two,three

// Array 리터럴을 사용합니다 .
var myArray:Array = ["one", "two", "three"];
trace(myArray); // 출력 : one,two,three
```

Array 클래스에는 인덱스 배열을 수정할 수 있는 속성 및 메서드도 포함되어 있습니다. 이 속성 및 메서드는 거의 모든 연관 배열에는 적용되지 않고 인덱스 배열에만 적용됩니다.

인덱스 배열은 인덱스 번호에 부호 없는 32비트 정수를 사용합니다. 인덱스 배열의 최대 크기는 2^{32} - 1 또는 4,294,967,295입니다. 배열의 크기가 최대 크기를 초과할 경우 런타임 오류가 발생합니다.

배열 요소에는 모든 유형의 데이터 값을 저장할 수 있습니다. 하지만 ActionScript 3.0에서는 *유형 배열* 개념이 지원되지 않습니다. 즉, 배열의 모든 요소가 특정 데이터 유형에 속하는 것으로 지정할 수 없습니다.

이 단원에서는 배열을 만드는 방법부터 Array 클래스를 사용하여 인덱스 배열을 만들고 수정하는 방법에 대해 설명합니다. 배열을 수정하는 메서드는 요소 삽입, 요소 제거 및 배열 정렬 방법의 세 가지 범주로 분류됩니다. 마지막 범주의 메서드는 기존 배열을 읽기 전용으로 처리하며 이 메서드는 단순히 배열에 대한 쿼리만 수행합니다. 즉, 기존 배열을 수정하지 않고 쿼리 메서드에서 모두 새 배열을 반환합니다. 이 단원의 마무리 부분은 Array 클래스 확장에 대해 다루고 있습니다.

배열 만들기

Array 생성자 함수는 세 가지 방법으로 사용할 수 있습니다. 첫 번째, 인수 없이 생성자를 호출할 경우 빈 배열을 얻습니다. Array 클래스의 `length` 속성을 사용하여 배열에 요소가 없는지 확인할 수 있습니다. 예를 들어, 다음 코드는 인수 없이 Array 생성자를 호출합니다.

```
var names:Array = new Array();
trace(names.length); // 출력 : 0
```

두 번째, Array 생성자에 대한 유일한 매개 변수로 숫자를 사용할 경우, 이 숫자에 해당하는 길이의 배열이 만들어지고 각 요소의 값은 `undefined`로 설정됩니다. 인수는 0과 4,294,967,295 사이의 부호 없는 정수여야 합니다. 예를 들어, 다음 코드는 단일 숫자 인수로 Array 생성자를 호출합니다.

```
var names:Array = new Array(3);
trace(names.length); // 출력 : 3
trace(names[0]); // 출력 : undefined
trace(names[1]); // 출력 : undefined
trace(names[2]); // 출력 : undefined
```

세 번째, 생성자를 호출하고 매개 변수로 요소 목록을 전달할 경우 각 매개 변수에 해당하는 요소로 배열이 만들어집니다. 다음 코드는 Array 생성자에 세 가지 인수를 전달합니다.

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // 출력 : 3
trace(names[0]); // 출력 : John
trace(names[1]); // 출력 : Jane
trace(names[2]); // 출력 : David
```

배열 리터럴이나 객체 리터럴을 사용하여 배열을 만들 수도 있습니다. 배열 리터럴은 다음 예제와 같이 배열 변수로 바로 할당될 수 있습니다.

```
var names:Array = ["John", "Jane", "David"];
```

배열 요소 삽입

`push()`, `unshift()` 및 `splice()`의 세 가지 Array 클래스 메서드를 사용하여 배열에 요소를 삽입할 수 있습니다. `push()` 메서드는 배열 끝에 하나 이상의 요소를 추가합니다. 즉, `push()` 메서드를 사용하여 배열에 삽입된 마지막 요소는 최상위 인덱스 번호를 가지게 됩니다. `unshift()` 메서드는 배열 시작 부분에 하나 이상의 요소를 삽입하며 이 위치의 인덱스 번호는 항상 0입니다. `splice()` 메서드는 지정된 인덱스 위치의 모든 항목을 배열에 삽입합니다.

다음 예제는 이 세 가지 메서드를 모두 보여 줍니다. planets 배열은 태양에 가까운 순서로 행성의 이름을 저장하기 위해 만들어진 배열입니다. 먼저, push() 메서드가 첫 항목 Mars를 추가하기 위해 호출됩니다. 그 다음, unshift() 메서드가 배열 맨 앞에 있어야 할 항목인 Mercury를 삽입하기 위해 호출됩니다. 마지막으로, splice() 메서드가 Mercury 뒤 및 Mars 앞에 Venus와 Earth 항목을 삽입하기 위해 호출됩니다. splice()에 전달된 첫 번째 인수인 정수 1은 인덱스 1에서 삽입을 시작하도록 지시합니다. splice()로 전달된 두 번째 인수인 정수 0은 삭제해야 할 항목이 없음을 의미합니다. 마지막으로 splice()에 전달된 세 번째 및 네 번째 인수인 Venus 및 Earth는 삽입할 항목입니다.

```
var planets:Array = new Array();
planets.push("Mars");        // 배열 내용 : Mars
planets.unshift("Mercury"); // 배열 내용 : Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets);              // 배열 내용 : Mercury,Venus,Earth,Mars
```

push() 및 unshift() 메서드는 모두 수정된 배열의 길이를 나타내는 부호 없는 정수를 반환합니다. splice() 메서드는 요소를 삽입하기 위해 사용될 때 빈 배열을 반환합니다. 이상해 보이지만 splice() 메서드의 다양한 용도를 생각해 보면 이해할 수 있습니다. splice() 메서드를 사용하여 배열에 요소를 삽입할 수 있을 뿐만 아니라 배열에서 요소를 제거할 수도 있습니다. 요소를 제거하기 위해 사용할 경우 splice() 메서드는 제거된 요소를 포함하는 배열을 반환합니다.

배열 요소 제거

pop(), shift() 및 splice()의 세 가지 Array 클래스 메서드를 사용하여 배열에서 요소를 제거할 수 있습니다. pop() 메서드는 배열 끝에서 요소를 제거합니다. 즉 최상위 인덱스 번호 위치에 있는 요소를 제거합니다. shift() 메서드는 배열 시작 부분에서 요소를 제거합니다. 즉, 항상 인덱스 번호 0의 위치에 있는 요소를 제거합니다. 요소를 삽입하는 데에도 사용할 수 있는 splice() 메서드는 이 메서드에 전달된 첫 번째 인수에 의해 지정된 인덱스 번호부터 시작하여 임의의 개수의 요소를 제거합니다.

다음 예제는 이 세 가지 메서드를 모두 사용하여 배열에서 요소를 제거합니다. oceans 배열은 거대한 수역의 이름을 저장하기 위해 만들어진 배열입니다. 배열 중 일부 이름은 바다가 아닌 호수라서 제거해야 합니다.

먼저, splice() 메서드를 사용하여 Aral 및 Superior 항목을 제거하고 Atlantic 및 Indian 항목을 삽입합니다. splice()에 전달된 첫 번째 인수인 정수 2는 인덱스 2 위치에 있는 목록의 세 번째 항목으로 작업을 시작해야 함을 나타냅니다. 두 번째 인수 2는 두 개의 항목이 제거되어야 함을 나타냅니다. 남은 인수인 Atlantic 및 Indian은 인덱스 2 위치에 삽입할 값입니다.

그 다음, pop() 메서드를 사용하여 배열에서 마지막 요소인 Huron을 제거합니다. 세 번째로 shift() 메서드를 사용하여 배열의 첫 번째 항목인 Victoria를 제거합니다.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian",
  "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // Aral 및 Superior를 바꿉니다.
oceans.pop();   // Huron을 제거합니다.
oceans.shift(); // Victoria를 제거합니다.
trace(oceans);  // 출력: Pacific,Arctic,Atlantic,Indian
```

pop() 및 shift() 메서드는 모두 제거된 항목을 반환합니다. 배열에 모든 데이터 유형의 값을 저장할 수 있으므로 반환 값의 데이터 유형은 Object입니다. splice() 메서드는 제거된 값을 포함하는 배열을 반환합니다. oceans 배열 예제를 변경하여, 다음 예제와 같이 splice()에 대한 호출에서 배열이 새 배열 변수로 할당되도록 할 수 있습니다.

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // 출력: Aral,Superior
```

배열 요소에서 delete 연산자를 사용하는 코드가 있을 수 있습니다. delete 연산자는 배열 요소 값을 undefined로 설정하지만 배열에서 요소를 제거하지는 않습니다. 예를 들어, 다음 코드는 oceans 배열의 세 번째 요소에서 delete 연산자를 사용하지만 배열의 길이는 여전히 5입니다.

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);        // 출력: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]);     // 출력: undefined
trace(oceans.length); // 출력: 5
```

배열의 length 속성을 사용하여 배열을 자를 수 있습니다. 배열의 length 속성을 현재 배열의 길이보다 작게 설정할 경우, 배열이 잘리고 새 length 값에서 1을 뺀 값보다 높은 인덱스 번호 위치에 저장된 모든 요소가 제거됩니다. 예를 들어, 다음 코드와 같이 oceans 배열을 배열 시작 부분에 모든 유효 항목이 오도록 정렬한 경우 length 속성을 사용하여 배열 끝에 있는 항목을 제거할 수 있습니다.

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // 출력: Arctic,Pacific
```

배열 정렬

reverse(), sort() 및 sortOn()의 세 가지 메서드를 사용하여 배열을 정렬 또는 역정렬하여 순서를 변경할 수 있습니다. 이 세 가지 메서드는 모두 기존 배열을 수정합니다. reverse() 메서드는 마지막 요소가 첫 번째 요소가 되게 하고 끝에서 두 번째 요소가 두 번째 요소가 되게 하는 등 배열 순서를 변경할 수 있습니다. sort() 메서드를 사용하면 사전 정의된 다양한 방법으로 배열을 정렬할 수 있으며 사용자 정의 정렬 알고리즘도 만들 수 있습니다.
sortOn() 메서드를 사용하면 정렬 키로 사용할 수 있는 하나 이상의 공통 속성이 있는 객체의 인덱스 배열을 정렬할 수 있습니다.

reverse() 메서드는 매개 변수를 취하지 않으며 값을 반환하지도 않습니다. 다만 이 메서드를 사용하여 배열의 순서를 현재 상태에서 역순으로 전환할 수 있습니다. 다음 예제는 oceans 배열에 나열된 바다의 순서를 역순으로 전환합니다.

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // 출력 : Pacific,Indian,Atlantic,Arctic
```

sort() 메서드는 *기본 정렬 순서*를 사용하여 배열의 요소를 다시 정렬합니다. 기본 정렬 순서의 특징은 다음과 같습니다.

- 대/소문자를 구분하며 대문자는 소문자보다 우선합니다. 예를 들어, 글자 D가 글자 b보다 우선합니다.
- 오름차순으로 낮은 문자 코드(예: A)가 높은 문자 코드(예: B)보다 우선합니다.
- 동일한 값은 특정 순서 없이 인접하여 배치합니다.
- 문자열 기반이므로, 요소를 비교하기 전에 문자열로 변환합니다. 예를 들어, 문자열 "1"이 문자열 "3"보다 낮은 문자 코드를 가지므로, 10이 3보다 먼저 오게 됩니다.

대/소문자를 구분하지 않고 배열을 정렬하거나 내림차순으로 정렬하거나 또는 숫자가 포함된 배열을 알파벳순이 아닌 숫자순으로 정렬해야 할 경우가 있습니다. sort() 메서드에는 기본 정렬 순서의 각 특성을 변경할 수 있는 options 매개 변수가 있습니다. 이 옵션은 다음 목록과 같이 Array 클래스의 정적 상수 세트로 정의됩니다.

- Array.CASEINSENSITIVE: 이 옵션은 정렬 시 대/소문자가 무시되도록 합니다. 예를 들어, 소문자 글자 b가 대문자 글자 D보다 우선합니다.
- Array.DESCENDING: 이 옵션은 기본 오름차순 정렬을 역으로 설정합니다. 예를 들어, 글자 B가 글자 A보다 우선합니다.
- Array.UNIQUESORT: 이 옵션은 동일한 값이 두 개 있을 경우 정렬을 중단하도록 합니다.
- Array.NUMERIC: 숫자순으로 정렬하는 옵션이므로 3이 10보다 먼저 오게 됩니다.

다음 예제는 이 옵션 중 일부를 보여 줍니다. poets라는 배열이 만들어져 다양한 여러 옵션을 사용하여 정렬됩니다.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // 기본 정렬
trace(poets); // 출력 : Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // 출력 : Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // 출력 : cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // 두 가지 옵션을 사용합
  니다 .
trace(poets); // 출력 : Dante,cummings,Blake,Angelou
```

사용자 정의 정렬 함수를 작성하여 이 함수를 sort() 메서드에 매개 변수로 전달할 수도 있습니다. 예를 들어, 각 목록 요소에 개인의 전체 이름이 포함된 이름 목록이 있는 경우, 이 목록을 성(姓)을 사용하여 정렬하려고 하면 사용자 정의 정렬 함수를 사용하여 각 요소를 파싱하고 정렬 함수에서 성을 사용해야 합니다. 다음 코드는 Array.sort() 메서드에 대해 사용자 정의 함수를 매개 변수로 사용하여 이 작업을 수행하는 방법을 보여 줍니다.

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
  var lastName:RegExp = /\b\S+$/;
  var name1 = a.match(lastName);
  var name2 = b.match(lastName);
  if (name1 < name2)
  {
    return -1;
  }
  else if (name1 > name2)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}

trace(names); // 출력 : John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // 출력 : Jane Doe,Mike Jones,John Q. Smith
```

사용자 정의 정렬 함수 orderLastName()은 일반 표현식을 사용하여 각 요소에서 비교 작업에 사용할 성을 추출합니다. 함수 식별자 orderLastName은 names 배열에서 sort() 메서드를 호출할 때 유일한 매개 변수로 사용됩니다. 정렬 함수는 한 번에 두 개의 배열 요소로 작업하므로 두 개의 매개 변수 a 및 b를 취합니다. 정렬 함수의 반환 값은 요소가 정렬되는 방법을 나타냅니다.

- 반환 값 -1은 첫 번째 매개 변수인 a가 두 번째 매개 변수인 b보다 우선함을 나타냅니다.
- 반환 값 1은 두 번째 매개 변수인 b가 첫 번째 매개 변수인 a보다 우선함을 나타냅니다.
- 반환 값 0은 요소의 정렬 우선 순위가 동일함을 나타냅니다.

sortOn() 메서드는 요소에 객체가 포함된 인덱스 배열을 위해 설계되었습니다. 이러한 객체에는 정렬 키로 사용할 수 있는 공통 속성이 최소한 하나는 있어야 합니다. 다른 유형의 배열에 sortOn() 메서드를 사용할 경우 예기치 못한 결과를 초래할 수 있습니다.

다음 예제는 각 요소가 문자열이 아닌 객체가 되도록 poets 배열을 수정합니다. 각 객체에는 시인의 성과 출생 연도가 포함되어 있습니다.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
```

```
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

sortOn() 메서드를 사용하여 born 속성으로 배열을 정렬할 수 있습니다. sortOn() 메서드는 두 개의 매개 변수 fieldName 및 options를 정의합니다. fieldName 인수는 문자열로 지정해야 합니다. 다음 예제에서는 두 개의 인수 "born" 및 Array.NUMERIC으로 sortOn()이 호출됩니다. Array.NUMERIC 인수는 알파벳순이 아닌 숫자순으로 정렬하려고 할 때 사용합니다. 이는 모든 숫자의 자릿수가 같은 경우에도 유용한 방법입니다. 나중에 이 배열에 자릿수가 많거나 적은 숫자를 추가하게 되더라도 예상한 대로 정렬이 수행될 수 있기 때문입니다.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
  trace(poets[i].name, poets[i].born);
}
/* 출력 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

일반적으로 sort() 및 sortOn() 메서드는 배열을 수정합니다. 기존 배열을 수정하지 않고 배열을 정렬하려면 Array.RETURNINDEXEDARRAY 상수를 options 매개 변수의 일부로 전달합니다. 이렇게 하면 메서드에서 정렬이 적용된 새 배열을 반환하고 원래의 배열은 수정하지 않고 남겨 둡니다. 메서드에서 반환된 배열은 새 정렬 순서가 적용된 간단한 인덱스 번호의 배열이며 원래 배열의 요소는 전혀 포함하지 않습니다. 예를 들어, 배열을 수정하지 않고 출생 연도순으로 poets 배열을 정렬하려면 Array.RETURNINDEXEDARRAY 상수를 options 매개 변수로 전달되는 인수의 일부로 포함합니다.

다음 예제에서는 반환된 인덱스 정보를 indices라는 배열에 저장하고, 이 indices 배열을 수정되지 않은 poets 배열과 함께 사용하여 출생 연도순으로 시인의 목록을 출력합니다.

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
  var index:int = indices[i];
  trace(poets[index].name, poets[index].born);
}
/* 출력 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
```

```
Angelou 1928
*/
```

배열 쿼리

Array 클래스의 남은 네 가지 메서드 concat(), join(), slice() 및 toString()은 모두 정보를 얻기 위해 배열에 쿼리를 실행하지만 배열을 수정하지는 않습니다. join() 및 toString() 메서드가 모두 문자열을 반환하는 반면 concat() 및 slice() 메서드는 모두 새 배열을 반환합니다. concat() 메서드는 인수로 요소 목록이나 새 배열을 취하며 이를 기존 배열과 결합하여 새 배열을 만듭니다. slice() 메서드에는 startIndex와 endIndex라는 이름의 매개 변수 두 개가 있으며 기존 배열에서 "분할된" 요소의 복사본이 포함된 새로운 배열을 반환합니다. 분할은 startIndex 위치의 요소에서 시작되어 endIndex 직전 위치에 있는 요소에서 종료됩니다. 즉, endIndex 위치의 요소는 반환 값에 포함되지 않습니다.

다음 예제에서는 다른 배열의 요소를 사용하여 새 배열을 만들기 위해 concat() 및 slice()를 사용합니다.

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // 출력 : alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // 출력 : alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // 출력 : alpha,beta,gamma
```

join() 및 toString() 메서드를 사용하여 배열에 쿼리를 실행하고 그 내용을 문자열로 반환할 수 있습니다. join() 메서드에 매개 변수를 사용하지 않는 경우 두 메서드는 동일하게 동작합니다. 즉, 배열에 있는 모든 요소의 목록이 쉼표로 구분되어 포함된 문자열을 반환합니다. toString() 메서드와는 달리 join() 메서드는 delimiter라는 매개 변수를 취하여, 반환된 문자열에서 각 요소 간을 분리하는 데 사용되는 구분 기호를 선택할 수 있습니다.

다음 예제에서는 rivers라는 배열을 만들고 join() 및 toString()을 모두 호출하여 배열의 값을 문자열로 반환합니다. toString() 메서드는 쉼표로 구분된 값(riverCSV)을 반환하는 데 사용되는 반면, join() 메서드는 + 문자로 구분된 값을 반환하는 데 사용됩니다.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // 출력 : Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // 출력 : Nile+Amazon+Yangtze+Mississippi
```

단, 다음 예제와 같이 중첩 배열에서는 `join()` 메서드에서 기본 배열 요소에 지정한 분리 기호와 관계없이 항상 쉼표로 구분된 값이 반환됩니다.

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // 출력 : a+b,c,d+e
```

연관 배열

해시 또는 *맵*이라고도 하는 연관 배열은 저장된 값을 구성하기 위해 숫자 인덱스를 사용하지 않고 *키*를 사용합니다. 연관 배열의 각 키는 저장된 값에 액세스하기 위해 사용되는 고유한 문자열입니다. 연관 배열은 Object 클래스의 인스턴스이며 각 키는 속성 이름에 해당합니다. 연관 배열은 정렬되지 않은 키/값 쌍 모음입니다. 연관 배열의 키에 특정한 순서가 지정되어 있는 것은 아닙니다.

ActionScript 3.0에는 *사전*이라는 고급 연관 배열 유형이 도입되었습니다. 사전은 flash.utils 패키지에 있는 Dictionary 클래스의 인스턴스이며 키를 사용합니다. 이 키는 모든 데이터 유형이 될 수 있지만 일반적으로는 Object 클래스의 인스턴스입니다. 즉, 사전 키는 String 유형 값 이외의 값도 가능합니다.

이 단원에서는 문자열을 키로 사용하는 연관 배열을 만드는 방법 및 Dictionary 클래스를 사용하는 방법에 대해 설명합니다.

문자열 키가 있는 연관 배열

ActionScript 3.0에서는 두 가지 방법으로 연관 배열을 만들 수 있습니다. 첫 번째 방법은 Object 생성자를 사용하는 것이며, 이 방법을 사용하면 객체 리터럴로 배열을 초기화할 수 있습니다. *일반 객체*라고도 하는 Object 클래스의 인스턴스는 연관 배열과 기능면에서 동일합니다. 일반 객체의 각 속성 이름은 저장된 값에 대한 액세스를 제공하는 키 역할을 합니다.

다음 예제는 `monitorInfo`라는 연관 배열을 만들고, 객체 리터럴을 사용하여 두 개의 키/값 쌍으로 배열을 초기화합니다.

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// 출력 : Flat Panel 1600 x 1200
```

선언 시 배열을 초기화할 필요가 없을 경우에는 다음과 같이 Object 생성자를 사용하여 배열을 만들 수 있습니다.

```
var monitorInfo:Object = new Object();
```

객체 리터럴이나 Object 클래스 생성자를 사용하여 배열을 만든 후에는 대괄호 연산자(`[]`)나 도트 연산자(`.`)를 사용하여 배열에 새 값을 추가할 수 있습니다. 다음 예제에서는 두 개의 새로운 값을 `monitorArray`에 추가합니다.

```
monitorInfo["aspect ratio"] = "16:10"; // 잘못된 형식입니다 . 공백을 사용하지 마십시
  오 .
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// 출력 : 16:10 16.7 million
```

aspect ratio 키에는 공백 문자가 포함됩니다. 이는 대문자 연산자를 사용하는 경우에만 가능하고 도트 연산자를 사용하는 경우에 공백 문자가 포함되면 오류가 발생합니다. 그러나 키 이름에 공백을 사용하는 것은 권장되지 않습니다.

연관 배열을 만드는 두 번째 방법은 Array 생성자를 사용한 다음 대괄호 연산자([])나 도트 연산자(.)를 사용하여 배열에 키/값 쌍을 추가하는 것입니다. 연관 배열을 Array 유형으로 선언할 경우에는 객체 리터럴을 사용하여 배열을 초기화할 수 없습니다. 다음 예제에서는 Array 생성자를 사용하여 monitorInfo라는 연관 배열을 만들고 type이라는 키와 resolution이라는 키를 각각의 값과 함께 추가합니다.

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// 출력 :  Flat Panel 1600 x 1200
```

Array 생성자를 사용하여 연관 배열을 만들어도 별다른 이점은 없습니다. Array 생성자나 Array 데이터 유형을 사용하는 경우에도 연관 배열에 Array.length 속성이나 Array 클래스의 메서드를 모두 사용할 수 없습니다. Array 생성자는 인덱스 배열을 만드는 데 사용하는 것이 가장 좋습니다.

객체 키가 있는 연관 배열

Dictionary 클래스를 사용하여 문자열이 아닌 객체를 키로 사용하는 연관 배열을 생성할 수 있습니다. 이러한 배열을 사전, 해시 또는 맵이라고도 합니다. 예를 들어, 특정 컨테이너와의 연관성을 기준으로 Sprite 객체의 위치를 결정하는 응용 프로그램을 살펴봅니다. 이 경우 Dictionary 객체를 사용하여 컨테이너에 각 Sprite 객체를 매핑할 수 있습니다.

다음 코드는 Dictionary 객체의 키 역할을 수행할 Sprite 클래스의 세 가지 인스턴스를 만듭니다. 각 키에는 GroupA 또는 GroupB의 값이 할당됩니다. 값은 모든 데이터 유형이 될 수 있지만 이 예제에서는 GroupA 및 GroupB가 모두 Object 클래스의 인스턴스입니다. 그런 후 다음 코드와 같이 속성 액세스([]) 연산자를 사용하여 각 키와 연관된 값에 액세스할 수 있습니다.

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// 키로 사용할 객체
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
```

```
var spr3:Sprite = new Sprite();

// 값으로 사용할 객체
var groupA:Object = new Object();
var groupB:Object = new Object();

// 사전에 새 키 - 값 쌍을 만듭니다 .
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
  trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
  trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
  trace("spr3 is in groupB");
}
```

객체 키로 반복

for..in 루프나 for each..in 루프를 사용하여 Dictionary 객체의 내용을 반복할 수 있습니다. for..in 루프를 사용하면 키를 기준으로 반복할 수 있으며, for each..in 루프를 사용하면 각 키와 연관된 값을 기준으로 반복할 수 있습니다.

Dictionary 객체의 객체 키에 직접 액세스하려면 for..in 루프를 사용합니다. 속성 액세스([]) 연산자를 사용하여 Dictionary 객체의 값에 액세스할 수도 있습니다. 다음 코드에서는 groupMap 사전의 이전 예제를 사용하여 for..in 루프를 통해 Dictionary 객체를 반복하는 방법을 보여 줍니다.

```
for (var key:Object in groupMap)
{
  trace(key, groupMap[key]);
}
/* 출력 :
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Dictionary 객체의 값에 직접 액세스하려면 for each..in 루프를 사용합니다. 다음 코드에서는 groupMap 사전을 사용하여 for each..in 루프를 통해 Dictionary 객체를 반복하는 방법을 보여 줍니다.

```
for each (var item:Object in groupMap)
```

```
{
  trace(item);
}
/* 출력 :
[object Object]
[object Object]
[object Object]
*/
```

객체 키 및 메모리 관리

Adobe Flash Player에서는 가비지 컬렉션 시스템을 사용하여 더 이상 사용되지 않는 메모리를 복구합니다. 객체에 객체를 가리키는 참조가 없을 경우 객체는 가비지 컬렉션의 대상이 되며 다음 번 가비지 컬렉션 시스템 실행 시 메모리가 복구됩니다. 예를 들어, 다음 코드는 새 객체를 만들어 변수 `myObject`에 객체에 대한 참조를 할당합니다.

```
var myObject:Object = new Object();
```

객체에 대한 참조가 남아 있는 한, 가비지 컬렉션 시스템에서는 객체가 사용하고 있는 메모리를 복구하지 않습니다. 다른 객체를 가리키거나 `null` 값으로 설정되는 등 `myObject` 값이 변경되면, 원래 객체에 의해 사용되던 메모리는 가비지 컬렉션의 대상이 됩니다. 그러나 이 원래 객체에 대한 다른 참조가 없는 경우에만 해당됩니다.

Dictionary 객체에서 `myObject`를 키로 사용하는 경우 원래 객체에 대한 다른 참조가 만들어집니다. 예를 들어, 다음 코드에서는 객체에 대한 두 개의 참조 즉, `myObject` 변수 및 `myMap` 객체의 키를 만듭니다.

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

`myObject`로 참조되는 객체를 가비지 컬렉션 대상으로 만들려면 객체에 대한 참조를 모두 제거해야 합니다. 이 경우, 다음 코드와 같이 `myObject` 값을 변경하고 `myMap`에서 `myObject` 키를 삭제해야 합니다.

```
myObject = null;
delete myMap[myObject];
```

또는 Dictionary 생성자의 `useWeakReference` 매개 변수를 사용하여 사전 키를 모두 *약한 참조*로 만들 수 있습니다. 가비지 컬렉션 시스템에서는 약한 참조를 무시하므로, 약한 참조만 있는 객체는 가비지 컬렉션의 대상이 됩니다. 예를 들어, 다음 코드에서는 객체를 가비지 컬렉션 대상으로 만들기 위해 `myMap`에서 `myObject` 키를 삭제할 필요가 없습니다.

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
```

```
myMap[myObject] = "foo";
myObject = null; // 객체를 가비지 컬렉션에 사용할 수 있게 만듭니다 .
```

다차원 배열

다차원 배열에는 다른 배열이 요소로 포함됩니다. 예를 들어, 인덱스 문자열 배열로 저장된 작업 목록을 살펴봅니다.

```
var tasks:Array = ["wash dishes", "take out trash"];
```

요일별로 별도의 작업 목록을 저장하려는 경우 각 요일별로 하나의 요소를 가진 다차원 배열을 만들 수 있습니다. 각 요소에는 작업 목록을 저장하는 `tasks` 배열과 비슷한 인덱스 배열이 포함됩니다. 다차원 배열에서는 인덱스 배열이나 연관 배열을 조합하여 사용할 수 있습니다. 다음 단원의 예제에서는 두 개의 인덱스 배열이나 인덱스 배열의 연관 배열을 사용합니다. 필요한 경우 다른 조합을 시도해 볼 수도 있습니다.

두 개의 인덱스 배열

두 개의 인덱스 배열을 사용할 경우 결과를 표나 스프레드시트로 표시할 수 있습니다.
첫 번째 배열의 요소는 표의 행을 나타내며 두 번째 배열의 요소는 열을 나타냅니다.

예를 들어, 다음 다차원 배열에서는 두 개의 인덱스 배열을 사용하여 요일별로 작업 목록을 추적합니다. 첫 번째 배열인 `masterTaskList`는 Array 클래스 생성자를 사용하여 만들어집니다. 배열의 각 요소는 요일을 나타내며 인덱스 0은 월요일, 인덱스 6은 일요일을 나타냅니다. 이 요소를 표의 행이라고 생각할 수 있습니다. `masterTaskList` 배열에서 만든 7개의 요소에 대해 각각 배열 리터럴을 할당하여 각 요일의 작업 목록을 만들 수 있습니다. 배열 리터럴은 표의 열을 나타냅니다.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

대괄호 표기법을 사용하여 모든 작업 목록에서 개별 항목에 액세스할 수 있습니다. 첫 번째 대괄호 세트는 요일을 나타내며 두 번째 대괄호 세트는 해당 요일에 대한 작업 목록을 나타냅니다. 예를 들어, 수요일 목록에서 두 번째 작업을 검색하려면, 먼저 수요일에 해당하는 인덱스 2를 사용한 다음 목록에서 두 번째 작업을 나타내는 인덱스 1을 사용합니다.

```
trace(masterTaskList[2][1]); // 출력 : dentist
```

일요일 목록에서 첫 번째 작업을 검색하려면 일요일에 해당하는 인덱스 6을 사용하고 목록에서 첫 번째 작업을 나타내는 인덱스 0을 사용합니다.

```
trace(masterTaskList[6][0]); // 출력 : mow lawn
```

인덱스 배열을 포함한 연관 배열

개별 배열에 보다 쉽게 액세스할 수 있도록 하기 위해, 요일에는 연관 배열을 사용하고 작업 목록에는 인덱스 배열을 사용할 수 있습니다. 연관 배열을 사용하면 특정 요일을 참조할 때 도트 구문을 사용할 수 있지만, 이 경우 연관 배열의 각 요소에 액세스하기 위한 런타임 처리 시간이 길어집니다. 다음 예제에서는 작업 목록의 기본으로 요일별 키/값 쌍이 있는 연관 배열을 사용합니다.

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

도트 구문을 사용하면 여러 개의 대괄호 세트를 사용하지 않아도 되므로 코드가 한결 읽기 쉬어집니다.

```
trace(masterTaskList.Wednesday[1]); // 출력 : dentist
trace(masterTaskList.Sunday[0]);    // 출력 : mow lawn
```

`for..in` 루프를 사용하여 작업 목록을 반복할 수 있지만 각 키와 연관된 값에 액세스하려면 도트 구문이 아닌 대괄호 표기법을 사용해야 합니다. `masterTaskList`는 연관 배열이기 때문에, 다음 예제와 같이 요소가 원하는 순서로 검색되지 않습니다.

```
for (var day:String in masterTaskList)
{
  trace(day + ": " + masterTaskList[day])
}
/* 출력 :
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

배열 복제

Array 클래스에는 배열의 복사본을 만들기 위한 메서드가 내장되어 있지 않습니다. 인수 없이 `concat()` 또는 `slice()` 메서드를 호출하여 배열의 *단순 복사본*을 만들 수 있습니다. 단순 복사본에서 원래 배열에 객체인 요소가 있는 경우 객체는 복사되지 않고 객체에 대한 참조만 복사됩니다. 복사본은 원래 배열과 동일한 객체를 가리킵니다. 객체에 대한 모든 변경 사항은 두 배열에 모두 반영됩니다.

*전체 복사본*에서는 원래 배열에 있던 객체도 모두 복사되어 새 배열이 원래 배열과 동일한 객체를 가리키지 않습니다. 전체 복사본을 작성하려면 둘 이상의 코드 행이 있어야 하며 일반적으로 함수를 작성해야 합니다. 이 함수는 전역 유틸리티 함수나 Array 하위 클래스의 메서드로 작성될 수 있습니다.

다음 예제에서는 전체 복사본을 작성하는 `clone()` 함수를 정의합니다. 알고리즘은 일반적인 Java 프로그래밍 기술을 사용합니다. 이 함수는 배열을 ByteArray 클래스 인스턴스로 직렬화하고 배열을 새 배열로 다시 읽어 전체 복사본을 만듭니다. 이 함수는 객체를 허용하므로 다음 코드와 같이 인덱스 배열 및 연관 배열 모두에 사용할 수 있습니다.

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
  var myBA:ByteArray = new ByteArray();
  myBA.writeObject(source);
  myBA.position = 0;
  return(myBA.readObject());
}
```

고급 항목

Array 클래스 확장

Array 클래스는 최종이 아닌 몇 개의 핵심 클래스 중 하나이므로, 사용자 고유의 Array 하위 클래스를 만들 수 있습니다. 이 단원에서는 Array 하위 클래스를 만드는 방법에 대한 예제와 이 과정 동안 발생할 수 있는 일부 문제에 대해 살펴봅니다.

앞서 언급했듯이 ActionScript에서 배열의 유형은 정의되어 있지 않지만 특정 데이터 유형의 요소만 허용하는 Array 하위 클래스를 만들 수 있습니다. 다음 단원의 예제에서는 요소를 첫 번째 매개 변수에 지정된 데이터 유형 값으로 제한하는 TypedArray라는 Array 하위 클래스를 정의합니다. TypedArray 클래스는 Array 클래스를 확장하는 방법을 설명하는 예제로만 사용되며 다음과 같은 여러 가지 이유로 실제로 사용하기에는 적합하지 않을 수 있습니다. 첫 번째, 컴파일 타임이 아닌 런타임에 유형 검사가 실행됩니다. 두 번째, TypedArray 메서드에서 불일치가 발생할 경우, 불일치가 무시되고 예외가 발생하지 않습니다. 메서드를 간단히 수정하여 예외가 발생되도록 할 수는 있습니다. 세 번째, 이 클래스에서는 배열에 모든 유형의 값을 삽입할 수 있는 배열 액세스 연산자의 사용을 막을 수 없습니다. 네 번째, 이 코딩 스타일에서는 성능 최적화보다 간결성을 선호합니다.

하위 클래스 선언

`extends` 키워드를 사용하여 클래스가 Array의 하위 클래스임을 나타낼 수 있습니다. Array 하위 클래스는 Array 클래스와 같이 `dynamic` 특성을 사용해야 합니다. 그렇지 않으면 하위 클래스가 올바르게 작동하지 않습니다.

다음 코드에서는 TypedArray 클래스의 정의를 보여 줍니다. 이 클래스에는 데이터 유형, 생성자 메서드 및 배열에 요소를 추가할 수 있는 네 가지 메서드를 보유할 수 있는 상수가 포함됩니다. 이 예제에서는 각 메서드의 코드가 생략되어 있지만 다음 단원에서 모두 자세히 살펴봅니다.

```
public dynamic class TypedArray extends Array
{
  private const dataType:Class;

  public function TypedArray(...args) {}

  AS3 override function concat(...args):Array {}

  AS3 override function push(...args):uint {}

  AS3 override function splice(...args) {}

  AS3 override function unshift(...args):uint {}
}
```

이 예제에서는 컴파일러 옵션 -as3이 true로 설정되어 있고 컴파일러 옵션 -es가 false로 설정되어 있다고 가정하므로 네 가지 대체 메서드에서는 모두 public 특성이 아닌 AS3 네임스페이스를 사용합니다. 이는 Adobe Flex Builder 2 및 Adobe Flash CS3 Professional의 기본 설정입니다. 자세한 내용은 170페이지의 "AS3 네임스페이스"를 참조하십시오.

팁 프로토타입 상속 사용을 선호하는 고급 개발자일 경우 TypedArray 클래스에서 두 가지 사항을 약간 변경하여 컴파일러 옵션 -es를 true로 설정하여 컴파일할 수 있습니다. 첫 번째, override 특성의 모든 항목을 제거하고 AS3 네임스페이스를 public 특성으로 바꿉니다. 두 번째, 네 가지 super 항목을 모두 Array.prototype으로 교체합니다.

TypedArray 생성자

생성자에서 임의의 길이의 인수 목록을 허용해야 하므로 하위 클래스 생성자에서 이와 관련한 문제가 발생할 수 있습니다. 문제는 배열을 만들기 위해 superconstructor로 인수를 전달하는 방법입니다. 인수 목록을 배열로 전달할 경우 superconstructor는 이것을 Array 유형의 단일 인수로 생각하여 결과 배열은 항상 1 요소 길이가 됩니다. 인수 목록 전달을 처리하는 기존의 방법은 Function.apply() 메서드를 사용하는 것이며, 이 메서드는 인수 배열을 두 번째 매개 변수로 취하지만 함수 실행 시에는 이것을 인수 목록으로 변환합니다. 그러나 Function.apply() 메서드는 생성자와 함께 사용할 수 없습니다.

남은 유일한 옵션은 TypedArray 생성자에서 Array 생성자의 논리를 다시 만드는 것입니다. 다음 코드는 Array 클래스 생성자에서 사용되는 알고리즘을 보여 줍니다. 이 알고리즘은 Array 하위 클래스 생성자에서 재사용할 수 있습니다.

```
public dynamic class Array
{
  public function Array(...args)
  {
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen;
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")");
      }
      length = ulen;
    }
    else
    {
      length = n;
      for (var i:int=0; i < n; i++)
      {
        this[i] = args[i]
      }
```

```
    }
  }
}
```

TypedArray 생성자는 다음과 같은 네 가지 변경 사항을 제외하고는 Array 생성자의 코드 대부분을 공유합니다. 첫 번째, 매개 변수 목록에 배열 데이터 유형의 지정을 허용하는 Class 유형의 새로운 필수 매개 변수가 포함됩니다. 두 번째, 생성자로 전달되는 데이터 유형이 `dataType` 변수에 할당됩니다. 세 번째, `else` 문에서 `length` 속성 값이 `for` 루프 뒤에 할당되어 `length`에는 올바른 유형의 인수만 포함됩니다. 네 번째, `for` 루프 본문은 `push()` 메서드의 대체 버전을 사용하므로 올바른 데이터 유형의 인수만 배열에 추가됩니다. 다음 예제에서는 TypedArray 생성자 함수를 보여 줍니다.

```
public dynamic class TypedArray extends Array
{
  private var dataType:Class;
  public function TypedArray(typeParam:Class, ...args)
  {
    dataType = typeParam;
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")")
      }
      length = ulen;
    }
    else
    {
      for (var i:int=0; i < n; i++)
      {
        // push() 에서 유형 검사 수행
        this.push(args[i])
      }
      length = this.length;
    }
  }
}
```

TypedArray 대체 메서드

TypedArray 클래스는 배열에 요소를 추가할 수 있는 네 가지 Array 클래스 메서드를 대체합니다. 각각의 경우 대체 메서드는 정확하지 않은 데이터 유형이 요소가 추가되는 것을 막기 위한 유형 검사를 추가합니다. 이렇게 하면 각 메서드가 해당 수퍼 클래스 버전을 호출합니다.

`push()` 메서드는 `for..in` 루프를 사용하여 인수 목록을 반복하며 각 인수에서 유형 검사를 수행합니다. 정확하지 않은 유형의 인수는 모두 `splice()` 메서드를 사용하여 `args` 배열에서 제거됩니다. `for..in` 루프가 종료되면 `args` 배열에는 `dataType` 유형 값만 포함됩니다. 그런 후 다음 코드와 같이 업데이트된 `args` 배열로 `push()`의 수퍼 클래스 버전이 호출됩니다.

```
AS3 override function push(...args):uint
{
  for (var i:* in args)
  {
    if (!(args[i] is dataType))
    {
      args.splice(i,1);
    }
  }
  return (super.push.apply(this, args));
}
```

`concat()` 메서드는 `passArgs`라는 임시 TypedArray를 만들어 유형 검사를 통과한 인수를 저장합니다. 이를 통해 `push()` 메서드에 있는 유형 검사 코드를 재사용할 수 있습니다. `for..in` 루프는 `args` 배열을 반복하며 각 인수에서 `push()`를 호출합니다. `passArgs`는 TypedArray 유형이므로 `push()`의 TypedArray 버전이 실행됩니다. 그러면 `concat()` 메서드가 다음 코드와 같이 해당 수퍼 클래스 버전을 호출합니다.

```
AS3 override function concat(...args):Array
{
  var passArgs:TypedArray = new TypedArray(dataType);
  for (var i:* in args)
  {
    // push()에서 유형 검사 수행
    passArgs.push(args[i]);
  }
  return (super.concat.apply(this, passArgs));
}
```

`splice()` 메서드는 임의의 인수 목록을 취하지만 처음 두 개의 인수는 항상 삭제할 인덱스 번호 및 요소의 수를 나타냅니다. 이것이 바로 대체 `splice()` 메서드가 인덱스 위치 2 이상의 `args` 배열 요소에 대해서만 유형을 검사하는 이유입니다. 코드에서 눈 여겨 볼 것은 `for` 루프 내에서 `splice()`에 대한 재귀 호출이 있는 것처럼 보이지만, `args`가 TypedArray가 아닌 Array 유형이므로 실제로는 재귀 호출이 아니라는 점입니다. 즉, `args.splice()`에 대한 호출은 메서드의 수퍼 클래스 버전에 대한 호출입니다. `for..in` 루프가 끝나면 다음 코드와 같이 `args` 배열에는 인덱스 위치 2 이상의 정확한 유형의 값만 포함되며 `splice()`는 해당 수퍼 클래스 버전을 호출합니다.

```
  AS3 override function splice(...args):*
  {
    if (args.length > 2)
    {
      for (var i:int=2; i< args.length; i++)
      {
        if (!(args[i] is dataType))
        {
          args.splice(i,1);
        }
      }
    }
    return (super.splice.apply(this, args));
  }
```

unshift() 메서드는 배열의 시작 부분에 요소를 추가하며 임의의 인수 목록을 허용합니다. 대체 unshift() 메서드는 다음 예제 코드와 같이 push() 메서드에서 사용한 것과 매우 유사한 알고리즘을 사용합니다.

```
  AS3 override function unshift(...args):uint
  {
    for (var i:* in args)
    {
      if (!(args[i] is dataType))
      {
        args.splice(i,1);
      }
    }
    return (super.unshift.apply(this, args));
  }
}
```

예제: PlayList

PlayList 예제는 노래 목록을 관리하는 음악 재생 목록 응용 프로그램에서 배열을 사용한 작업에 관련된 기술에 대해 설명합니다. 이러한 기술은 다음과 같습니다.

- 인덱스 배열 만들기
- 인덱스 배열에 항목 추가
- 다른 정렬 옵션을 사용하여 다른 속성별로 객체 배열 정렬
- 문자 구분 문자열로 배열 변환

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. PlayList 응용 프로그램 파일은 Samples/PlayList 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
PlayList.mxml 또는 PlayList.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/playlist/Song.as	노래 한 곡에 대한 정보를 나타내는 값 객체. PlayList 클래스에서 관리되는 항목은 Song 인스턴스입니다.
com/example/programmingas3/playlist/SortProperty.as	Song 객체 목록을 정렬할 수 있는 Song 클래스 속성을 나타내는 사용 가능한 값의 의사 열거 항목입니다.

PlayList 클래스 개요

PlayList 클래스는 Song 객체 세트를 관리하며, 재생 목록에 노래를 추가(`addSong()` 메서드)하고 목록에서 노래를 정렬(`sortList()` 메서드)할 수 있는 기능을 제공하는 공용 메서드를 가지고 있습니다. 또한 이 클래스에는 읽기 전용 접근자 속성인 `songList`가 포함되며, 이 속성을 통해 재생 목록에 있는 노래 세트에 실제로 액세스할 수 있습니다. 내부적으로 PlayList 클래스는 전용 Array 변수를 사용하여 노래를 추적합니다.

```
public class PlayList
{
  private var _songs:Array;
  private var _currentSort:SortProperty = null;
  private var _needToSort:Boolean = false;
  ...
}
```

노래 목록을 추적하기 위해 PlayList 클래스에서 사용되는 _songs Array 변수 외에도 두 개의 다른 전용 변수를 통해 목록을 정렬해야 하는지 여부(_needToSort) 및 지정된 시간에 정렬해야 하는 노래 목록의 속성(_currentSort)을 추적할 수 있습니다.

다른 객체와 마찬가지로 Array 인스턴스를 선언하는 것은 Array를 만드는 작업에서 반 정도를 완료한 것이며, Array 인스턴스의 속성이나 메서드에 액세스하려면 먼저 PlayList 클래스의 생성자에서 이를 인스턴스화해야 합니다.

```
public function PlayList()
{
  this._songs = new Array();
  // 초기 정렬을 설정합니다 .
  this.sortList(SortProperty.TITLE);
}
```

생성자의 첫 번째 행에서 _songs 변수를 인스턴스화하면 사용 준비가 완료됩니다. 또한 초기 정렬 기준 속성을 설정할 수 있는 sortList() 메서드가 호출됩니다.

목록에 노래 추가

사용자가 응용 프로그램에 새로운 노래를 넣으면 데이터 입력 양식의 코드가 PlayList 클래스의 addSong() 메서드를 호출합니다.

```
/**
 * 재생 목록에 노래를 추가합니다 .
 */
public function addSong(song:Song):void
{
  this._songs.push(song);
  this._needToSort = true;
}
```

addSong() 내에서 _songs 배열의 push() 메서드가 호출되어 addSong()에 전달된 Song 객체가 해당 배열의 새 요소로 추가됩니다. push() 메서드를 사용하면 이전에 적용된 정렬에 관계없이 새 요소가 배열의 끝에 추가됩니다. 즉, push() 메서드가 호출된 후에는 노래 목록이 제대로 정렬되어 있지 않게 되므로 _needToSort 변수가 true로 설정됩니다. 이론적으로는 목록이 지정된 시간에 정렬되었는지 여부를 추적할 필요 없이 sortList() 메서드를 바로 호출할 수 있습니다. 하지만 실제로는 노래 목록을 검색하기 바로 전까지는 노래 목록을 정렬할 필요가 없습니다. 이렇게 정렬 작업을 지연하면, 노래를 검색하기 전에 여러 노래가 목록에 추가되는 경우 등에 응용 프로그램에서 불필요한 정렬을 수행하지 않아도 됩니다.

노래 목록 정렬

재생 목록에서 관리되는 Song 인스턴스는 복잡한 객체이므로, 응용 프로그램 사용자가 재생 목록을 노래 제목이나 발표 연도와 같은 다른 속성에 따라 정렬하고 싶어할 수 있습니다. PlayList 응용 프로그램에서 노래 목록을 정렬하는 작업은 세 가지 부분으로 나눌 수 있습니다. 첫 번째는 목록을 정렬해야 하는 속성을 식별하는 것이며 두 번째는 해당 속성별로 정렬할 때 사용해야 하는 정렬 옵션을 지정하는 것이며 세 번째는 실제로 정렬 작업을 수행하는 것입니다.

정렬 속성

Song 객체는 노래 제목, 아티스트, 발표 연도, 파일 이름 및 사용자가 선택한 노래 장르 등을 비롯한 여러 속성을 추적합니다. 이 중, 처음 세 가지만 정렬에 사용할 수 있습니다. 개발자의 편의에 따라 이 예제에는 정렬 시 사용 가능한 속성을 나타내는 값이 나열된 SortProperty 클래스가 포함됩니다.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

SortProperty 클래스에는 세 가지 상수 즉, TITLE, ARTIST 및 YEAR가 포함되어 있으며, 이러한 상수는 각각 정렬 시 사용할 수 있는 연관된 Song 클래스 속성의 실제 이름이 포함된 문자열을 저장합니다. 나머지 코드에서 정렬 속성이 지정될 때마다 이 열거 항목을 사용하여 정렬이 수행됩니다. 예를 들어, PlayList 생성자에서 목록은 초기에는 다음과 같이 sortList() 메서드를 호출하여 정렬됩니다.

```
// 초기 정렬을 설정합니다 .
this.sortList(SortProperty.TITLE);
```

정렬 속성이 SortProperty.TITLE로 지정되므로 노래가 제목에 따라 정렬됩니다.

속성순 정렬 및 정렬 옵션 지정

노래 목록을 실제로 정렬하는 작업은 다음과 같이 sortList() 메서드의 PlayList 클래스에 의해 수행됩니다.

```
/**
 * 지정된 속성에 따라 노래 목록을 정렬합니다 .
 */
public function sortList(sortProperty:SortProperty):void
{
  ...
  var sortOptions:uint;
  switch (sortProperty)
  {
    case SortProperty.TITLE:
      sortOptions = Array.CASEINSENSITIVE;
```

```
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // 데이터의 실제 정렬을 수행합니다 .
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // 현재 정렬 속성을 저장합니다 .
    this._currentSort = sortProperty;

    // 목록이 정렬되었음을 기록합니다 .
    this._needToSort = false;
  }
```

제목이나 아티스트순으로 정렬할 경우 알파벳순으로 정렬하는 것이 좋은 방법이지만 연도순으로 정렬할 경우에는 숫자 정렬이 가장 적합합니다. `switch` 문은 적합한 정렬 옵션을 정의하는 데 사용되며 `sortProperty` 매개 변수에 지정된 값에 따라 `sortOptions` 변수에 저장됩니다. 여기서 다시 하드 코딩된 값이 아닌 지정된 열거 항목이 속성을 구별하는 데 사용됩니다.

결정된 정렬 속성 및 정렬 옵션을 사용하여 `sortOn()` 메서드를 호출하고 매개 변수로 두 개의 값을 전달하면 `_songs` 배열이 실제로 정렬됩니다. 그러면 현재 정렬된 노래 목록에 따라 현재 정렬 속성이 기록됩니다.

문자 구분 문자열로 배열 요소 결합

PlayList 클래스의 노래 목록을 관리하기 위해 배열이 사용되는 것 이외에도 이 예제에서는 지정된 노래가 속한 장르 목록을 관리하기 위해 Song 클래스에서도 배열이 사용됩니다. Song 클래스 정의에서 다음 코드를 살펴보십시오.

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint,
  filename:String, genres:Array)
{
  ...
  // 장르가 배열로 전달되지만
  // 세미콜론으로 구분된 문자열로 저장됩니다 .
  this._genres = genres.join(";");
}
```

새로운 Song 인스턴스를 만들 때 노래가 속한 장르를 지정하는 데 사용되는 genres 매개 변수는 Array 인스턴스로 정의됩니다. 이렇게 하면 여러 장르를 생성자로 전달할 수 있는 하나의 변수로 묶을 때 편리합니다. 하지만 내부적으로 Song 클래스는 전용 _genres 변수에서 장르를 세미콜론으로 구분된 문자열 인스턴스로 관리합니다. Array 매개 변수는 리터럴 문자열 값 ";"을 지정된 구분 기호로 하여 join() 메서드를 호출함으로써 세미콜론으로 구분된 문자열로 변환됩니다.

마찬가지로 genres 접근자를 통해 장르를 Array로 설정하거나 검색할 수 있습니다.

```
public function get genres():Array
{
  // 장르가 세미콜론으로 구분된 String으로 저장되므로
  // 장르를 Array로 변형해야 Array로부터 다시 전달할 수 있습니다.
  return this._genres.split(";");
}
public function set genres(value:Array):void
{
  // 장르가 배열로 전달되지만
  // 세미콜론으로 구분된 문자열로 저장됩니다.
  this._genres = value.join(";");
}
```

genres set 접근자는 생성자와 정확히 동일하게 동작합니다. 즉, Array를 허용하며 join() 메서드를 호출하여 이것을 세미콜론으로 구분된 문자열로 변환합니다. 반면 get 접근자는 이와 반대의 작업을 수행합니다. 즉 _genres 변수의 split() 메서드를 호출하여 지정된 구분 기호(앞서 언급한 리터럴 문자열 값인 ";")를 사용하여 String을 값의 배열로 분할합니다.

제 8 장

8 오류 처리

오류를 "처리"한다는 것은 응용 프로그램 컴파일 시 또는 컴파일된 응용 프로그램 실행 시 생성된 오류에 응답하거나 이를 수정하는 논리를 응용 프로그램 내에 작성하는 것을 의미합니다. 응용 프로그램에서 오류를 처리할 때는 응답이 없거나 오류가 발생한 프로세스가 자동으로 실패되는 것과 달리 오류가 발생하면 *특정 작업*이 수행됩니다. 정확하게 사용한다면 오류 처리를 통해 예기치 못한 비헤이비어로부터 응용 프로그램 및 사용자를 보호할 수 있습니다.

하지만 오류 처리는 컴파일이나 런타임 시 발생하는 수많은 오류에 대한 응답을 비롯한 광범위한 범주를 포함하고 있습니다. 이 장에서는 런타임 오류 처리 방법, 생성될 수 있는 다양한 오류 유형 및 ActionScript 3.0에서 제공되는 새로운 오류 처리 시스템의 장점에 대해 중점적으로 살펴봅니다. 또한 응용 프로그램에 대한 사용자 정의 오류 처리 전략의 구현 방법에 대해서도 설명합니다.

목차

오류 처리의 기초

오류 처리 소개

런타임 오류는 ActionScript 코드에 잘못된 데이터가 들어갈 경우 발생하며 Adobe Flash Player에서 ActionScript 내용이 실행되는 것을 중단시킵니다. ActionScript 코드가 올바르게 실행되도록 하려면 응용 프로그램 내에서 오류를 처리하는 코드를 작성해야 합니다. 이 코드는 오류를 수정 및 해결하거나 적어도 사용자에게 오류가 발생했음을 알리는 역할을 합니다. 이 프로세스가 바로 *오류 처리* 프로세스입니다.

오류 처리는 컴파일이나 런타임 시 발생하는 수많은 오류에 대한 응답 등이 포함된 광범위한 범주입니다. 컴파일 작업 시 발생하는 오류는 쉽게 확인되는 경우가 많습니다. SWF 파일 생성 프로세스를 완료하기 위해서는 이 오류를 수정해야 합니다. 이 장에서는 컴파일 타임 오류에 대해서는 다루지 않습니다. 컴파일 타임 오류를 포함하지 않는 코드 작성에 대한 자세한 내용은 63페이지의 제3장, “ActionScript 언어 및 구문” 및 131페이지의 제4장, “ActionScript의 객체 지향 프로그래밍”을 참조하십시오. 이 장에서는 런타임 오류에 대해 중점적으로 설명합니다.

런타임 오류는 오류 코드를 실제로 실행할 경우에만 발생하기 때문에 감지하기가 쉽지 않습니다. `if..then..else` 명령문에서처럼 프로그램의 세그먼트에 코드의 분기가 여러 개일 경우, 코드에 오류가 없음을 확인하기 위해서는 실제 사용자가 입력할 수 있는 모든 가능한 입력 값을 사용하여 가능한 모든 조건을 테스트해보아야 합니다.

런타임 오류는 두 가지 범주로 구분할 수 있습니다. *프로그램 오류*는 메서드 매개 변수에 잘못된 데이터 유형을 지정하는 것과 같은 ActionScript 코드 내 오류를 뜻하며, *논리 오류*는 자금 관리용 프로그램에서 잘못된 공식으로 이율을 계산하는 경우 등의 프로그램의 논리(데이터 확인 및 값 조작) 오류입니다. 다시 말하지만, 이 두 유형의 오류 모두 대개 응용 프로그램을 꼼꼼히 테스트함으로써 사전에 감지하고 수정할 수 있습니다.

가장 이상적인 것은 최종 사용자에게 응용 프로그램을 릴리스하기 전에 모든 오류를 식별하여 제거하는 것이지만, 모든 오류를 사전에 발견하거나 예방하는 것은 불가능합니다. 예를 들어, ActionScript 응용 프로그램이 프로그래머가 통제할 수 없는 특정 웹사이트로부터 정보를 불러온다고 가정해 보겠습니다. 이 경우 특정 시점에 해당 웹사이트에 접근할 수 없으면 외부 데이터에 의존하고 있는 응용 프로그램의 일부가 제대로 동작하지 않을 수 있습니다. 가장 중요한 오류 처리 과정에는 사용자가 응용 프로그램을 계속해서 사용할 수 있도록 하거나, 적어도 제대로 동작하지 않는 이유를 설명하는 친숙한 오류 메시지가 표시되도록 이와 같은 알 수 없는 상황에 대한 해결 방법을 준비하고 적절하게 처리하는 작업이 포함됩니다.

런타임 오류는 ActionScript에서 다음과 같은 두 가지 방법으로 나타납니다.

- 오류 클래스: 상당수의 오류는 해당 오류와 연관된 오류 클래스가 있습니다. 오류가 발생하면 Flash Player에서 해당 오류와 연관된 특정 오류 클래스의 인스턴스를 만듭니다. 그러면 코드에서 오류 객체에 포함된 정보를 확인한 후 해당 오류에 대해 적절히 응답합니다.

- 오류 이벤트: Flash Player에서 정상적으로 이벤트를 트리거하는 경우에도 오류가 발생할 때가 있습니다. 이러한 경우 Flash Player에서는 오류 이벤트를 대신 트리거합니다. 다른 이벤트와 마찬가지로 각 오류 이벤트에는 연관된 클래스가 있으며, Flash Player는 해당 클래스의 인스턴스를 오류 이벤트를 구독하는 메서드로 전달합니다.

특정 메서드에서 오류 또는 오류 이벤트를 트리거할 수 있는지 알아보려면 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 메서드 항목을 찾아보십시오.

일반적인 오류 처리 작업

다음은 코드 작성 시 수행해야 하는 일반적인 오류 관련 작업입니다.

- 오류 처리 코드 작성
- 테스트, 오류 catch 및 re-throw
- 사용자 정의 오류 클래스 정의
- 오류 및 상태 이벤트에 응답

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 비동기: 메서드 호출과 같이 즉각적인 결과를 제공하지 않는 프로그램 명령으로, 대신 결과(또는 오류)가 이벤트 형태로 제공됩니다.
- Catch: 예외(런타임 오류) 발생 시 코드에서 해당 예외를 인식하면 코드가 예외를 *catch*했다고 합니다. 예외가 포착되면 Flash Player에서 다른 ActionScript 코드로 예외를 알리는 작업이 중단됩니다.
- 디버거 버전: 사용자에게 런타임 오류를 알리는 코드가 포함된 특수 버전의 Flash Player입니다. 가장 많이 사용되는 Flash Player 표준 버전에서 ActionScript 코드에 의해 처리되지 않은 오류는 Flash Player에서 무시됩니다. 디버거 버전(Adobe Flash CS3 Professional 및 Adobe Flex와 함께 제공됨)에서는 처리되지 않은 오류가 발생할 경우 경고 메시지가 나타납니다.
- 예외: 프로그램이 실행되는 동안 발생되어 런타임 환경(즉, Flash Player)에서 자체적으로 해결할 수 없는 오류를 나타냅니다.
- Re-throw: 코드에서 예외를 포착하면 Flash Player에서는 예외를 더 이상 다른 객체에 알리지 않습니다. 그러나 예외를 다른 객체에 알리는 것이 중요할 경우 코드에서 해당 예외를 *re-throw*하여 알림 프로세스를 다시 시작해야 합니다.
- 동기: 즉각적인 결과가 제공되거나 오류를 즉시 throw하는 프로그램 명령(예: 메서드 호출)으로, 동일한 코드 블록 내에서 해당 응답을 사용할 수 있습니다.
- Throw: Flash Player를 비롯하여 결과적으로 다른 객체 및 ActionScript 코드에 오류 발생을 알리는 행위를 오류를 *throwing*한다고 합니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 기본적으로 이 장의 모든 코드 샘플에는 해당하는 `trace()` 함수 호출이 포함되어 있습니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널에서 코드 샘플의 `trace()` 함수에 대한 결과를 확인합니다.

이후의 코드 샘플 중 일부는 보다 복잡하며 클래스로 작성되어 있습니다. 이러한 예제를 테스트하려면:

1. 빈 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 새 ActionScript 파일을 만들고 Flash 문서와 같은 디렉토리에 저장합니다. 파일 이름은 코드 샘플에 있는 클래스 이름과 일치해야 합니다. 예를 들어, 코드 목록에 ErrorTest라는 클래스가 정의된 경우 ActionScript 파일을 ErrorTest.as라는 이름으로 저장합니다.
3. ActionScript 파일에 코드 샘플을 복사하고 파일을 저장합니다.
4. Flash 문서에서 스테이지 또는 작업 영역의 빈 부분을 클릭하여 문서 속성 관리자를 활성화합니다.
5. 텍스트에서 복사한 ActionScript 클래스의 이름을 속성 관리자의 [문서 클래스] 필드에 입력합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널(예제가 `trace()` 함수를 사용할 경우) 또는 예제 코드에 의해 작성된 텍스트 필드에서 예제의 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

오류 유형

응용 프로그램을 개발하고 실행할 때 다양한 오류 유형 및 오류 용어를 접하게 됩니다. 다음 목록은 주요 오류 유형 및 용어에 대한 설명입니다.

- *컴파일 시간 오류*: 코드 컴파일 중 ActionScript 컴파일러에서 발생합니다. 컴파일 시간 오류는 코드의 구문 문제로 인해 응용 프로그램을 제대로 빌드하지 못할 때 발생합니다.
- *런타임 오류*: 컴파일 후 응용 프로그램을 실행할 때 발생합니다. 런타임 오류는 SWF 파일이 Adobe Flash Player 9에서 재생될 때 발생하는 오류를 나타냅니다. 대부분의 경우 런타임 오류가 발생하면 사용자에게 오류 내용을 보고하고 응용 프로그램 실행을 유지하기 위한 절차를 수행하여 오류를 처리할 수 있습니다. 원격 웹사이트에 연결할 수 없거나 필수 데이터를 로드할 수 없는 등 치명적 오류일 경우 오류 처리를 사용하여 응용 프로그램을 적절하게 종료할 수 있습니다.
- *동기 오류*: 함수 호출 시 발생하는 런타임 오류입니다. 예를 들어, 특정 메서드를 사용하려고 할 때 이 메서드로 전달한 인수가 유효하지 않은 경우 Flash Player에서 예외가 발생합니다. 대부분의 오류는 명령문 실행 시 동기적으로 발생하며, 가장 적합한 `catch` 문으로 제어의 흐름이 즉시 전달됩니다.

  예를 들어, 다음 코드 예제에서는 프로그램에서 파일 업로드를 시도하기 전에 `browse()` 메서드가 호출되지 않아 런타임 오류가 발생합니다.

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
    fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
    trace(error);
    // 오류 #2037: 잘못된 순서로 함수가 호출되었거나 이전
    // 호출이 실패했습니다 .
  }
  ```

  이 경우 Flash Player에서 파일 업로드를 시도하기 전에 `browse()` 메서드가 호출되지 않았음을 확인하므로 런타임 오류가 동기적으로 발생합니다.

  동기 오류 처리에 대한 자세한 내용은 248페이지의 “응용 프로그램에서 동기 오류 처리”를 참조하십시오.
- *비동기 오류*: 런타임 중 다양한 지점에서 발생하는 런타임 오류입니다. 이 오류는 이벤트를 생성하며 이벤트 리스너에서 이를 포착합니다. 비동기 작업은 함수에서 작업을 시작하지만 완료될 때까지 기다리지 않는 작업입니다. 이때 오류 이벤트 리스너를 만들어 응용 프로그램이나 사용자가 특정 작업을 시도할 때까지 기다릴 수 있으며 작업에 실패할 경우 이벤트 리스너로 오류를 포착하여 오류 이벤트에 응답할 수 있습니다. 그러면 이벤트 리스너는 이벤트 핸들러 함수를 호출하여 오류 이벤트에 적절하게 응답합니다. 예를 들어, 이벤트 핸들러는 사용자에게 오류를 해결하도록 하는 대화 상자를 표시할 수 있습니다.

앞에서 설명했던 파일 업로드 동기 오류를 살펴봅니다. 파일 업로드를 시작하기 전에 `browse()` 메서드를 성공적으로 호출한 경우 Flash Player에서 여러 이벤트를 전달하게 됩니다. 예를 들어, 업로드가 시작되면 `open` 이벤트가 전달됩니다. 그리고 파일 업로드 작업이 성공적으로 완료되면 `complete` 이벤트가 전달됩니다. 이벤트 처리는 비동기적이므로(즉, 특정 시간, 알려진 시간 또는 미리 지정된 시간에 발생하지 않음) 다음 코드와 같이 이러한 특정 이벤트를 수신하려면 `addEventListener()` 메서드를 사용해야 합니다.

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
  trace("...select...");
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  fileupload.cfm");
  request.method = URLRequestMethod.POST;
  event.target.upload(request.url);
}
function openHandler(event:Event):void
{
  trace("...open...");
}
function completeHandler(event:Event):void
{
  trace("...complete...");
}
```

비동기 오류 처리에 대한 자세한 내용은 254페이지의 “오류 이벤트 및 상태에 응답”을 참조하십시오.

- *catch되지 않는 예외*: 오류에 응답하기 위한 해당 논리(예: `catch` 문)가 없을 때 발생하는 오류입니다. 응용 프로그램에서 오류가 발생하고 현재 또는 상위 수준에 오류를 처리할 수 있는 적합한 `catch` 문이나 이벤트 핸들러가 없는 경우 이 오류는 포착되지 않는 예외로 간주됩니다.

  런타임 시 Flash Player에서는 포착되지 않는 오류를 무시하고 오류가 발생해도 현재 SWF 파일이 중단되지 않는 경우 재생을 계속하도록 설계되어 있습니다. 이는 사용자가 오류를 자체 해결할 수 없기 때문입니다. catch되지 않는 오류를 무시하는 프로세스를 “자동 실패”라고 하며 자동 실패 시 디버깅 응용 프로그램이 복잡해질 수 있습니다. Flash Player의 디버거 버전은 현재 스크립트를 종료하고 `trace` 문 출력에 catch되지 않는 오류를 표시하거나 로그 파일에 오류 메시지를 기록함으로써 catch되지 않는 오류에 응답합니다. 예외 객체가 Error 클래스 또는 그 하위 클래스 중 하나의 인스턴스일 경우 `getStackTrace()` 메서드가 호출되며 스택 추적 정보도 trace 문 출력이나 로그 파일에 표시됩니다. Flash Player의 디버거 버전 사용에 대한 자세한 내용은 247페이지의 “Flash Player의 디버거 버전 작업”을 참조하십시오.

ActionScript 3.0에서 오류 처리

많은 응용 프로그램이 오류 처리를 위한 논리 작성 없이도 실행될 수 있으므로 개발자들은 응용 프로그램에 대한 오류 처리 관련 작업을 뒤로 미루는 경향이 있습니다. 하지만 오류 처리가 없으면 예상대로 작업이 수행되지 않을 경우 응용 프로그램이 쉽게 멈추거나 사용자 작업을 방해할 수 있습니다. ActionScript 2.0에는 사용자가 논리를 사용자 정의 함수에 작성하여 특정 메시지와 함께 예외를 발생시킬 수 있는 Error 클래스가 있습니다. 오류 처리는 사용자에게 친숙한 응용 프로그램을 만들기 위해 매우 중요하므로 ActionScript 3.0에는 오류를 포착하기 위한 확장된 아키텍처가 포함되어 있습니다.

예외

_ActionScript 3.0 언어 및 구성 요소 참조 설명서_에 많은 메서드에 의해 발생된 예외가 설명되어 있지만 각 메서드에서 발생할 수 있는 모든 예외가 포함된 것은 아닙니다. 메서드 설명 부분에 메서드에서 발생하는 일부 예외가 나열되어 있는 경우에도 이 설명에서 명확하게 언급하지 않은 다른 문제나 구문 오류에 의해 메서드에서 예외가 발생할 수 있습니다.

ActionScript 3.0 오류 처리 요소

ActionScript 3.0에는 다음을 비롯하여 오류 처리를 위한 여러 도구가 포함되어 있습니다.

- Error 클래스: ECMAScript(ECMA-262) Edition 4 초안 언어 사양에 따라 ActionScript 3.0에는 광범위한 Error 클래스가 포함되어 오류 객체를 생성할 수 있는 상황 범위를 확장합니다. 각 Error 클래스는 시스템 오류(예: MemoryError의 경우), 코딩 오류(예: ArgumentError의 경우), 네트워킹 및 통신 오류(예: URIError의 경우) 등의 특정 오류 또는 기타 상황을 응용 프로그램에서 처리하고 응답할 수 있도록 도와줍니다. 각 클래스에 대한 자세한 내용은 257페이지의 “오류 클래스 비교”를 참조하십시오.
- 자동 실패 감소: 이전 Flash Player 버전에서는 명시적으로 `throw` 문을 사용한 경우에만 오류가 생성되고 보고되었습니다. Flash Player 9의 경우 기본 ActionScript 메서드 및 속성에서, 예외 발생 시 보다 효율적으로 이를 처리하고 각 예외에 개별적으로 대응할 수 있도록 하기 위해 런타임 오류가 발생됩니다.
- 디버깅 시 명확한 오류 메시지 표시: Flash Player의 디버거 버전을 사용하는 경우 문제가 될 수 있는 코드나 상황에서 확실한 오류 메시지가 발생되며 이를 통해 특정 코드 블록이 실패하는 이유를 쉽게 파악할 수 있습니다. 따라서 오류를 보다 효율적으로 수정할 수 있습니다. 자세한 내용은 247페이지의 “Flash Player의 디버거 버전 작업”을 참조하십시오.
- 정확한 오류를 통해 런타임 시 사용자에게 명확한 오류 메시지 표시: 이전 버전의 Flash Player에서는 `upload()` 호출이 실패할 경우 `FileReference.upload()` 메서드에서 5가지 가능한 오류 중 하나를 나타내는 `false` 부울 값을 반환했습니다. ActionScript 3.0에서 `upload()` 메서드 호출 시 오류가 발생하면 4가지 특정 오류 중 하나를 발생시킬 수 있으며 이로 인해 최종 사용자에게 보다 정확한 오류 메시지를 표시할 수 있습니다.

- 개선된 오류 처리: 많은 공통적인 상황에서 서로 다른 오류가 발생합니다. 예를 들어, ActionScript 2.0에서는 FileReference 객체가 채워지기 전에는 `name` 속성이 `null` 값을 가지고 있습니다. 따라서 `name` 속성을 사용하거나 표시하기 전에, 값이 설정되었고 `null`이 아닌지 확인해야 합니다. ActionScript 3.0에서는 FileReference 객체가 채워지기 전에 `name` 속성에 액세스하려고 할 경우 Flash Player에서 IllegalOperationError를 발생시켜, 값이 설정되지 않았으며 이 오류를 처리하려면 `try..catch..finally` 블록을 사용할 수 있음을 알려줍니다. 자세한 내용은 248페이지의 "try..catch..finally 문 사용"을 참조하십시오.
- 성능상의 중대한 단점 없음: `try..catch..finally` 블록을 사용하면 이전 버전의 ActionScript에 비해 리소스를 거의 추가로 사용하지 않고도 오류를 처리할 수 있습니다.
- ErrorEvent 클래스를 통해 특정 비동기 오류 이벤트에 대한 리스너를 작성할 수 있습니다. 자세한 내용은 254페이지의 "오류 이벤트 및 상태에 응답"을 참조하십시오.

오류 처리 전략

응용 프로그램에 문제 상황이 발생하지 않는 한 오류 처리 논리를 코드에 작성하지 않아도 응용 프로그램을 성공적으로 실행할 수 있습니다. 하지만 오류를 적극적으로 처리하지 않고 응용 프로그램에 문제가 발생한 경우 사용자는 문제 발생 시 원인을 알 수 없습니다.

여러 가지 방법을 사용하여 응용 프로그램에서 오류를 처리할 수 있으며, 다음 목록에는 오류를 처리하는 세 가지 주요 옵션이 요약되어 있습니다.

- `try..catch..finally` 문 사용: 동기 오류 발생 시 오류를 포착합니다. 계층으로 명령문을 중첩하여 다양한 코드 실행 수준에서 예외를 포착할 수 있습니다. 자세한 내용은 248페이지의 "try..catch..finally 문 사용"을 참조하십시오.
- 고유한 사용자 정의 오류 객체 만들기: Error 클래스를 사용하여 고유의 사용자 정의 오류 객체를 만들어 내장된 오류 유형에 포함되지 않는 응용 프로그램의 특정 작업을 추적할 수 있습니다. 그런 다음 사용자 정의 오류 객체에 `try..catch..finally` 문을 사용할 수 있습니다. 자세한 내용은 253페이지의 "사용자 정의 오류 클래스 만들기"를 참조하십시오.
- 오류 이벤트에 응답할 이벤트 리스너 및 핸들러 작성: 이 전략을 사용하면 `try..catch..finally` 블록에서 많은 코드를 중복하지 않고도 유사한 이벤트를 처리할 수 있는 전역 오류 핸들러를 만들 수 있습니다. 이 전략을 사용하여 비동기 오류를 포착할 수도 있습니다. 자세한 내용은 254페이지의 "오류 이벤트 및 상태에 응답"을 참조하십시오.

Flash Player의 디버거 버전 작업

Adobe에서는 개발자의 디버깅 작업을 지원하기 위해 특별 버전의 Flash Player를 제공합니다. Adobe Flash CS3 Professional 또는 Adobe Flex Builder 2 설치 시 Flash Player의 디버거 버전을 사용할 수 있습니다.

Flash Player의 디버거 버전과 릴리스 버전에서 오류를 표시하는 방법에는 확연한 차이가 있습니다. 디버거 버전에는 오류 유형(예: 일반 오류, IOError 또는 EOFError), 오류 번호 및 사용자가 쉽게 이해할 수 있는 오류 메시지가 표시됩니다. 반면 릴리스 버전에는 오류 유형과 오류 번호만 표시됩니다. 예를 들어, 다음과 같은 코드를 살펴봅니다.

```
try
{
  tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
  tf.text = error.toString();
}
```

Flash Player의 디버거 버전에서는 `readBoolean()` 메서드에서 EOFError가 발생하면 `tf` 텍스트 필드에 "EOFError: Error #2030: End of file was encountered."라는 메시지가 표시됩니다.

Flash Player 릴리스 버전에서는 동일한 코드에서 "EOFError: Error #2030."라는 텍스트가 표시됩니다.

릴리스 버전에서는 Flash Player의 리소스 및 크기를 최소화하기 위해 오류 메시지 문자열을 표시하지 않습니다. 대신 설명서(*ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 부록)에서 오류 번호와 관련된 오류 메시지를 찾아볼 수 있습니다. 또는 전체 메시지를 확인하기 위해 Flash Player의 디버거 버전을 사용하여 오류를 다시 생성할 수 있습니다.

응용 프로그램에서 동기 오류 처리

가장 일반적인 오류 처리는 동기 오류 처리 논리이며, 런타임 시 동기 오류를 catch하기 위해 코드에 명령문을 삽입합니다. 이러한 유형의 오류 처리를 통해 함수가 실패할 경우 응용 프로그램에 런타임 오류를 알리고 복구할 수 있습니다. 동기 오류를 포착하기 위한 논리에는 `try..catch..finally` 문이 포함되며, 이는 실제로 작업을 다시 시도하여 Flash Player에서 오류 응답을 포착하고 마지막으로 실패한 작업을 처리하기 위해 일부 다른 작업을 실행합니다.

try..catch..finally 문 사용

동기 런타임 오류가 발생하면 `try..catch..finally` 문을 사용하여 오류를 포착합니다. 런타임 오류가 발생하면 Flash Player에서 예외가 발생되어 정상적인 실행이 중지되고 Error 유형의 특수 객체가 만들어집니다. 그런 다음 이 Error 객체는 첫 번째 사용 가능한 `catch` 블록에 전달됩니다.

`try` 문에는 오류가 발생할 가능성이 있는 명령문이 포함됩니다. 따라서 `try` 문에는 항상 `catch` 문을 함께 사용합니다. `try` 문 블록의 명령문 중 하나에서 오류가 감지되면 해당 `try` 문에 연결된 `catch` 문이 실행됩니다.

`finally` 문에는 `try` 블록에서 오류가 발생하는지 여부와 관계없이 실행되는 명령문이 포함됩니다. 오류가 없으면 `finally` 블록 내의 명령문이 `try` 블록 명령문이 완료된 후 실행됩니다. 오류가 있으면 먼저 적합한 `catch` 문이 실행되고 `finally` 블록의 명령문이 뒤따라 실행됩니다.

다음 코드에서는 `try..catch..finally` 문을 사용한 구문 예를 보여 줍니다.

```
try
{
   // 오류를 throw할 수 있는 코드
}
catch (err:Error)
{
   // 오류에 응답하는 코드
}
finally
{
   // 오류가 throw되었는지 여부에 관계없이 실행되는 코드. 이 코드는 오류 발생 후
   // 정리를 수행하거나 응용 프로그램을 계속 실행하기 위한 조치를 수행합니다.
}
```

각 catch 문은 명령문이 처리하는 특정 예외 유형을 식별합니다. catch 문은 Error 클래스의 하위 클래스인 오류 클래스만 지정할 수 있으며, 각 catch 문은 순서대로 검사됩니다. 오류 발생 유형과 일치하는 첫 번째 catch 문만 실행됩니다. 즉, 먼저 상위 수준의 Error 클래스를 검사한 다음 Error 클래스의 하위 클래스를 검사할 경우 상위 수준의 Error 클래스에서만 일치가 발생합니다. 다음 코드는 이런 상황에 대한 예입니다.

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
```

이전 코드는 다음을 출력합니다.

```
<Error> I am an ArgumentError
```

ArgumentError를 정확하게 catch하려면 다음 코드와 같이 가장 세부적인 오류 유형을 먼저 나열하고 보다 일반적인 오류 유형을 그 다음에 나열해야 합니다.

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
```

Flash Player API의 여러 메서드 및 속성은 실행 시 오류가 발생하면 런타임 오류를 발생시킵니다. 예를 들어, 다음 코드와 같이 메서드에서 오디오 스트림을 닫을 수 없는 경우 Sound 클래스의 close() 메서드에서 IOError가 발생합니다.

```
var mySound:Sound = new Sound();
try
{
  mySound.close();
}
catch (error:IOError)
{
```

```
  // 오류 #2029: 이 URLStream 객체에는 열려 있는 스트림이 없습니다.
}
```

*ActionScript 3.0 언어 및 구성 요소 참조 설명서*에 익숙해지면 각 메서드 설명에 자세히 나와 있듯이 어떤 메서드에서 예외가 발생하는지 확인할 수 있습니다.

throw 문

응용 프로그램에서 런타임 오류가 발생하면 Flash Player에서 예외가 발생합니다. 또한 사용자가 직접 `throw` 문을 사용하여 예외를 명시적으로 발생시킬 수 있습니다. 오류를 명시적으로 발생시킬 경우 Adobe에서는 Error 클래스나 그 하위 클래스의 인스턴스를 생성할 것을 권장합니다. 다음 코드는 오류가 발생한 후 응답하기 위해 Error 클래스 인스턴스인 `MyErr`를 생성하고 `myFunction()` 함수를 호출하는 `throw` 문을 보여 줍니다.

```
var MyError:Error = new Error("Encountered an error with the numUsers
  value", 99);
var numUsers:uint = 0;
try
{
  if (numUsers == 0)
  {
    trace("numUsers equals 0");
  }
}
catch (error:uint)
{
  throw MyError; // 부호 없는 정수 오류를 catch합니다.
}
catch (error:int)
{
  throw MyError; // 정수 오류를 catch합니다.
}
catch (error:Number)
{
  throw MyError; // 숫자 오류를 catch합니다.
}
catch (error:*)
{
  throw MyError; // 다른 모든 오류를 catch합니다.
}
finally
{
  myFunction(); // 여기에서 필요한 정리 작업을 수행합니다.
}
```

가장 세부적인 데이터 유형이 먼저 나열되도록 `catch` 문이 정렬되어 있습니다. Number 데이터 유형에 대한 `catch` 문이 먼저 나열된 경우 uint 데이터 유형에 대한 `catch` 문이나 int 데이터 유형에 대한 catch 문이 모두 실행되지 않습니다.

예외

Java 프로그래밍 언어에서는 예외를 발생시킬 수 있는 각 함수의 경우, 함수 선언에 연결된 `throws` 절에서 발생할 수 있는 예외 클래스를 나열하여 이를 선언해야 합니다. ActionScript에서는 함수에서 발생할 수 있는 예외를 선언하지 않아도 됩니다.

간단한 오류 메시지 표시

새로운 예외 및 오류 이벤트 모델의 가장 큰 장점 중 하나는 사용자에게 액션이 실패한 시기 및 이유를 알려줄 수 있도록 한다는 것입니다. 즉, 메시지를 표시할 코드를 작성하고 그에 따른 응답 옵션을 제공할 수 있습니다.

다음 코드는 텍스트 필드에 오류를 표시하기 위한 간단한 `try..catch` 문을 보여 줍니다.

```
package
{
  import flash.display.Sprite;
  import flash.text.TextField;

  public class SimpleError extends Sprite
  {
    public var employee:XML =
      <EmpCode>
        <costCenter>1234</costCenter>
        <costCenter>1-234</costCenter>
      </EmpCode>;

    public function SimpleError()
    {
      try
      {
        if (employee.costCenter.length() != 1)
        {
          throw new Error("Error, employee must have exactly one cost
  center assigned.");
        }
      }
      catch (error:Error)
      {
        var errorMessage:TextField = new TextField();
        errorMessage.autoSize = TextFieldAutoSize.LEFT;
        errorMessage.textColor = 0xFF0000;
        errorMessage.text = error.message;
        addChild(errorMessage);
      }
    }
```

```
    }
}
```

ActionScript 3.0은 다양한 오류 클래스 및 내장 컴파일러 오류를 사용하여 ActionScript의 이전 버전보다 작업 실패 이유에 대한 보다 자세한 정보를 제공합니다. 따라서 보다 나은 오류 처리를 통해 응용 프로그램을 안정적으로 작성할 수 있습니다.

오류 rethrow

응용 프로그램 작성 시 오류를 올바르게 처리할 수 없는 경우 오류를 rethrow해야 하는 상황이 있을 수 있습니다. 예를 들어, 다음 코드는 중첩된 `catch` 블록에서 오류를 처리할 수 없는 경우 사용자 정의 ApplicationError를 rethrow하는 중첩된 `try..catch` 블록을 보여 줍니다.

```
try
{
  try
  {
    trace("<< try >>");
    throw new ArgumentError("some error which will be rethrown");
  }
  catch (error:ApplicationError)
  {
    trace("<< catch >> " + error);
    trace("<< throw >>");
    throw error;
  }
  catch (error:Error)
  {
    trace("<< Error >> " + error);
  }
}
catch (error:ApplicationError)
{
  trace("<< catch >> " + error);
}
```

이전 코드의 출력은 다음과 같습니다.

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

중첩된 `try` 블록은 이후 `catch` 블록에서 catch한 사용자 정의 ApplicationError 오류를 발생시킵니다. 이 중첩된 `catch` 블록에서 오류를 처리하려고 시도할 수 있으며 실패할 경우 이 블록을 포함하고 있는 `try..catch` 블록에 ApplicationErrorblock 객체를 전달합니다.

사용자 정의 오류 클래스 만들기

ActionScript에서 표준 Error 클래스 중 하나를 확장하여 고유한 오류 클래스를 만들 수 있습니다. 고유한 오류 클래스를 만드는 데는 다음과 같은 여러 이유가 있습니다.

- 응용 프로그램에 고유한 특정 오류나 오류 그룹을 식별하기 위해 만듭니다.

  예를 들어, Flash Player에서 발생한 오류뿐만 아니라 고유한 코드에서 발생한 오류에 대해 다른 액션을 취하고 싶을 수 있습니다. 이 경우 Error 클래스의 하위 클래스를 만들어 `try..catch` 블록에서 새로운 오류 데이터 유형을 추적할 수 있습니다.
- 응용 프로그램에서 생성된 오류에 대해 고유한 오류 표시 기능을 제공하기 위해 만듭니다.

  예를 들어, 특정 방식으로 오류 메시지를 포맷하는 새로운 `toString()` 메서드를 만들 수 있습니다. 오류 코드를 사용하여 사용자 언어 환경 설정을 기준으로 올바른 메시지를 검색하는 `lookupErrorString()` 메서드를 정의할 수도 있습니다.

고유한 오류 메시지는 핵심 ActionScript Error 클래스를 확장해야 합니다. 다음은 Error 클래스를 확장하는 고유한 AppError 클래스에 대한 예제입니다.

```
public class AppError extends Error
{
  public function AppError(message:String, errorID:int)
  {
    super(message, errorID);
  }
}
```

다음은 프로젝트에서 AppError를 사용하는 예제를 보여 줍니다.

```
try
{
  throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
  trace(error.errorID + ": " + error.message)
}
```

예외

하위 클래스에서 `Error.toString()` 메서드를 대체하려면 하나의 . . .(rest) 매개 변수를 제공해야 합니다. ECMAScript(ECMA-262) 버전 3 언어 사양에서는 이런 방식으로 `Error.toString()` 메서드가 정의되며 ActionScript 3.0에서는 해당 사양에 대한 역호환성을 위해 이 메서드를 동일한 방식으로 정의합니다. 그러므로 `Error.toString()` 메서드를 대체할 경우 매개 변수를 정확하게 일치시켜야 합니다. 런타임에 `toString()` 메서드에 다른 매개 변수를 전달해도 이 매개 변수는 무시됩니다.

오류 이벤트 및 상태에 응답

ActionScript 3.0 오류 처리에서 가장 주목할 만한 개선 사항 중 하나는 비동기 런타임 오류에 응답하기 위한 오류 이벤트 처리가 지원된다는 것입니다. 비동기 오류에 대한 정의는 243페이지의 "오류 유형"을 참조하십시오.

이벤트 리스너 및 이벤트 핸들러를 만들어 오류 이벤트에 응답할 수 있습니다. 많은 클래스에서 다른 이벤트를 전달하는 것과 같은 방식으로 오류 이벤트를 전달합니다. 예를 들어, XMLSocket 클래스의 인스턴스는 일반적으로 `Event.CLOSE`, `Event.CONNECT`, `DataEvent.DATA`의 세 가지 유형의 이벤트를 전달합니다. 하지만 문제 발생 시 XMLSocket 클래스는 `IOErrorEvent.IOError` 또는 `SecurityErrorEvent.SECURITY_ERROR`를 전달할 수 있습니다. 이벤트 리스너 및 이벤트 핸들러에 대한 자세한 내용은 295페이지의 제10장, "이벤트 처리"를 참조하십시오.

오류 이벤트는 다음 두 범주 중 하나에 해당됩니다.

- ErrorEvent 클래스를 확장하는 오류 이벤트

  flash.events.ErrorEvent 클래스에는 네트워킹 및 통신 작업과 관련한 Flash Player 런타임 오류를 관리하기 위한 속성 및 메서드가 포함됩니다. AsyncErrorEvent, IOErrorEvent, SecurityErrorEvent 클래스는 ErrorEvent 클래스를 확장합니다. Flash Player의 디버거 버전을 사용하는 경우에는 런타임에 대화 상자가 표시되어 플레이어에서 발생하는 리스너 함수가 없는 모든 오류 이벤트를 알립니다.

- 상태 기반 오류 이벤트

  상태 기반 오류 이벤트는 네트워킹 및 통신 클래스의 `netStatus` 및 `status` 속성과 관련됩니다. Flash Player에서 데이터를 읽거나 쓸 때 문제가 발생하면 사용 중인 클래스 객체에 따라 `netStatus.info.level`이나 `status.level` 속성의 값이 `"error"` 값으로 설정됩니다. 이벤트 핸들러 함수에서 `level` 속성에 `"error"` 값이 포함되었는지 여부를 확인하여 이 오류에 응답합니다.

오류 이벤트를 사용한 작업

ErrorEvent 클래스 및 그 하위 클래스에는 데이터를 읽거나 쓸 때 Flash Player에서 전달한 오류 처리를 위한 오류 유형이 포함됩니다.

다음 예제는 `try..catch` 문 및 오류 이벤트 핸들러를 모두 사용하여 로컬 파일을 읽으려고 할 때 감지된 모든 오류를 표시합니다. 보다 정교한 처리 코드를 추가하여 사용자에게 옵션을 제공하거나 "여기에 오류 처리 코드를 입력하십시오." 주석이 표시된 위치에서 자동으로 오류를 처리할 수 있습니다.

```
package
{
  import flash.display.Sprite;
  import flash.errors.IOError;
```

```
  import flash.events.IOErrorEvent;
  import flash.events.TextEvent;
  import flash.media.Sound;
  import flash.media.SoundChannel;
  import flash.net.URLRequest;
  import flash.text.TextField;

  public class LinkEventExample extends Sprite
  {
    private var myMP3:Sound;
    public function LinkEventExample()
    {
      myMP3 = new Sound();
      var list:TextField = new TextField();
      list.autoSize = TextFieldAutoSize.LEFT;
      list.multiline = true;
      list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
      list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
      addEventListener(TextEvent.LINK, linkHandler);
      addChild(list);
    }

    private function playMP3(mp3:String):void
    {
      try
      {
        myMP3.load(new URLRequest(mp3));
        myMP3.play();
      }
      catch (err:Error)
      {
        trace(err.message);
        // 여기에 오류 처리 코드를 입력하십시오 .
      }
      myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
    }

    private function linkHandler(linkEvent:TextEvent):void
    {
      playMP3(linkEvent.text);
      // 여기에 오류 처리 코드를 입력하십시오 .
    }

    private function errorHandler(errorEvent:IOErrorEvent):void
    {
      trace(errorEvent.text);
      // 여기에 오류 처리 코드를 입력하십시오 .
    }
  }
}
```

상태 변경 이벤트를 사용한 작업

Flash Player는 `level` 속성을 지원하는 클래스에 대해 `netStatus.info.level`이나 `status.level` 속성 값을 동적으로 변경합니다. `netStatus.info.level` 속성이 있는 클래스는 NetConnection, NetStream 및 SharedObject이며, `status.level` 속성이 있는 클래스는 HTTPStatusEvent, Camera, Microphone 및 LocalConnection입니다. 핸들러 함수를 작성하여 `level` 값의 변경에 응답하고 통신 오류를 추적할 수 있습니다.

다음 예제는 `netStatusHandler()` 함수를 사용하여 `level` 속성의 값을 테스트합니다. `level` 속성이 오류가 발생했음을 나타내는 경우 코드에서 "Video stream failed" 메시지를 생성합니다.

```
package
{
  import flash.display.Sprite;
  import flash.events.NetStatusEvent;
  import flash.events.SecurityErrorEvent;
  import flash.media.Video;
  import flash.net.NetConnection;
  import flash.net.NetStream;

  public class VideoExample extends Sprite
  {
    private var videoUrl:String = "Video.flv";
    private var connection:NetConnection;
    private var stream:NetStream;

    public function VideoExample()
    {
      connection = new NetConnection();
      connection.addEventListener(NetStatusEvent.NET_STATUS,
  netStatusHandler);
      connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
  securityErrorHandler);
      connection.connect(null);
    }

    private function netStatusHandler(event:NetStatusEvent):void
    {
      if (event.info.level = "error")
      {
        trace("Video stream failed")
      }
      else
      {
        connectStream();
      }
    }
```

```
    private function securityErrorHandler(event:SecurityErrorEvent):void
    {
      trace("securityErrorHandler: " + event);
    }

    private function connectStream():void
    {
      var stream:NetStream = new NetStream(connection);
      var video:Video = new Video();
      video.attachNetStream(stream);
      stream.play(videoUrl);
      addChild(video);
    }
  }
}
```

오류 클래스 비교

ActionScript에는 여러 가지 미리 정의된 Error 클래스가 제공됩니다. 이러한 클래스의 대부분은 Flash Player에서 사용되지만 고유 코드에서도 동일한 Error 클래스를 사용할 수 있습니다. ActionScript 3.0에는 두 가지 주요 Error 클래스 즉, ActionScript 핵심 Error 클래스 및 flash.error 패키지 Error 클래스가 있습니다. 핵심 Error 클래스는 ECMAScript(ECMA-262) 버전 4 초안 언어 사양에서 규정됩니다. flash.error 패키지 내용은 ActionScript 3.0 응용 프로그램 개발 및 디버깅을 지원하기 위해 도입된 추가 클래스입니다.

ECMAScript 핵심 Error 클래스

ECMAScript 핵심 Error 클래스에는 Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError 및 URIError 클래스가 포함됩니다. 이 클래스는 각각 최상위 네임스페이스에 있습니다.

클래스 이름	설명	참고
Error	Error 클래스는 예외 발생 시 사용될 수 있으며 ECMAScript (EvalError, RangeError, ReferenceError, SyntaxError, TypeError 및 URIError)에서 정의된 다른 예외 클래스에 대한 기본 클래스입니다.	Error 클래스는 Flash Player에서 발생하는 모든 런타임 오류에 대해 기본 클래스 역할을 수행하며 모든 사용자 정의 오류 클래스에 대해 권장된 기본 클래스입니다.
EvalError	EvalError 예외는 매개 변수가 Function 클래스의 생성자에 전달되거나 사용자 코드에서 `eval()` 함수를 호출할 경우 발생합니다.	ActionScript 3.0에서는 `eval()` 함수에 대한 지원이 없어져 이 함수를 사용하려고 하면 오류가 발생합니다. 이전 버전의 Flash Player에서는 `eval()` 함수를 사용하여 변수, 속성, 객체 또는 무비 클립을 이름으로 액세스할 수 있었습니다.
RangeError	RangeError 예외는 숫자 값이 허용되는 범위를 벗어난 경우 발생합니다.	예를 들어, 지연 시간이 음수이거나 유한수가 아닌 경우 Timer 클래스에서 RangeError가 발생합니다. 잘못된 심도에 표시 객체를 추가하려는 경우에도 RangeError가 발생할 수 있습니다.
ReferenceError	ReferenceError 예외는 봉인된 비동적 객체에서 정의되지 않은 속성을 참조하려고 하는 경우 발생합니다. ActionScript 3.0 이전의 ActionScript 컴파일러 버전에서는 `undefined` 속성에 액세스하려고 할 때 오류가 발생하지 않았습니다. 하지만 새 ECMAScript 사양에서 이 상황에서 오류를 발생하도록 규정하므로 ActionScript 3.0에서는 ReferenceError 예외가 발생합니다.	정의되지 않은 변수에 대한 예외는 잠재적인 버그를 식별할 수 있도록 해주며 소프트웨어 품질을 향상시키는 데 도움이 됩니다. 하지만 변수를 초기화하는 데 익숙하지 않은 경우 이 새로운 ActionScript 비헤이비어로 인해 코딩 습관을 바꿔야 할 수 있습니다.

클래스 이름	설명	참고
SyntaxError	SyntaxError 예외는 ActionScript 코드에서 파싱 오류가 발생할 경우 발생합니다. 자세한 내용은 ECMAScript(ECMA-262) 버전 3(버전 4 출시 전까지) 언어 사양(www.ecma-international.org/publications/standards/Ecma-262.htm)의 15.11.6.4절 및 ECMAScript for XML(E4X) 사양(ECMA-357 버전 2)(www.ecma-international.org/publications/standards/Ecma-357.htm)의 10.3.1절을 참조하십시오.	SyntaxError는 다음과 같은 상황에서 발생할 수 있습니다. • RegExp 클래스에 의해 잘못된 일반 표현식이 파싱되는 경우 • XMLDocument 클래스에 의해 잘못된 XML이 파싱되는 경우
TypeError	피연산자의 실제 유형이 예상 유형과 다르면 TypeError 예외가 발생합니다. 자세한 내용은 ECMAScript 사양(www.ecma-international.org/publications/standards/Ecma-262.htm)의 15.11.6.5절 및 E4X 사양(www.ecma-international.org/publications/standards/Ecma-357.htm)의 10.3절을 참조하십시오.	TypeError는 다음과 같은 상황에서 발생할 수 있습니다. • 함수 또는 메서드의 실제 매개 변수를 형식 매개 변수 유형으로 강제 변환할 수 없는 경우 • 변수에 할당된 값을 해당 변수의 유형으로 강제 변환할 수 없는 경우 • `is` 또는 `instanceof` 연산자에서 오른쪽 객체의 유형이 잘못된 경우 • `super` 키워드를 잘못 사용한 경우 • 속성을 조회한 결과 여러 바인딩이 발견되어 결과가 모호한 경우 • 호환되지 않는 객체에 메서드를 호출한 경우. 예를 들어, RegExp 클래스의 메서드를 일반 객체로 “가져와” 호출하면 TypeError 예외가 발생합니다.
URIError	URIError 예외는 전역 URI 처리 함수 중 하나를 해당 정의와 호환되지 않는 방식으로 사용할 경우 발생합니다. 자세한 내용은 ECMAScript 사양(www.ecma-international.org/publications/standards/Ecma-262.htm)의 15.11.6.6절을 참조하십시오.	URIError는 다음과 같은 상황에서 발생할 수 있습니다. `Socket.connect()`와 같은 유효한 URI를 예상하는 Flash Player API 함수에 유효하지 않은 URI가 지정되는 경우

ActionScript 핵심 Error 클래스

ActionScript는 핵심 ECMAScript Error 클래스뿐만 아니라 ActionScript 특정 오류 상황 및 오류 처리 기능을 위해 여러 자체 클래스를 추가합니다.

이러한 클래스는 ECMAScript 버전 4 초안 언어 사양에 대한 ActionScript 언어 확장으로, 초안 사양에 추가될 가능성이 있으므로 flash.error와 같은 패키지가 아닌 최상위 수준에 있습니다.

클래스 이름	설명	참고
ArgumentError	ArgumentError 클래스는 함수 호출 시 제공된 매개 변수 값이 해당 함수에 대해 정의된 매개 변수와 일치하지 않을 때 발생하는 오류를 나타냅니다.	인수 오류는 다음과 같은 상황에서 발생할 수 있습니다. • 메서드에 제공된 인수가 너무 적거나 너무 많은 경우 • 인수가 열거된 항목에 속해야 하지만 아닌 경우
SecurityError	SecurityError 예외는 보안 위반이 발생하여 액세스가 거부될 경우 발생합니다.	보안 오류는 다음과 같은 상황에서 발생할 수 있습니다. • 보안 샌드박스 경계를 벗어나 무단으로 속성에 액세스하거나 메서드를 호출하는 경우 • 보안 샌드박스에서 허용되지 않는 URL에 액세스하는 경우 • 정책 파일이 없을 때 1024 미만 포트에 연결하는 경우와 같이 권한이 없는 포트 번호에 소켓 연결을 시도하는 경우 • 사용자의 카메라 또는 마이크에 액세스를 시도했지만 장치 액세스 요청을 사용자가 거부한 경우
VerifyError	VerifyError 예외는 형식이 잘못되었거나 손상된 SWF 파일이 발견되는 경우 발생합니다.	SWF 파일이 다른 SWF 파일을 로드할 때 부모 SWF는 로드된 SWF에서 발생한 VerifyError를 catch할 수 있습니다.

flash.error 패키지 Error 클래스

flash.error 패키지에는 Flash Player API의 일부인 Error 클래스가 포함됩니다. 바로 앞서 설명한 Error 클래스와 달리 flash.error 패키지는 Flash Player 특정 오류 이벤트를 전달합니다.

클래스 이름	설명	참고
EOFError	EOFError 예외는 사용 가능한 데이터의 끝 부분을 지나 계속 읽으려고 하면 발생합니다.	예를 들어, IDataInput 인터페이스에서 읽기 메서드 중 하나가 호출되고 이 읽기 요청을 수행하기 위한 데이터가 부족한 경우 EOFError가 발생합니다.
IllegalOperationError	IllegalOperationError 예외는 메서드를 구현하지 않았거나 현재의 사용이 구현 범위에서 벗어나는 경우 발생합니다.	잘못된 작업 오류 예외는 다음과 같은 상황에서 발생할 수 있습니다. • DisplayObjectContainer와 같은 기본 클래스에서 Stage에서 지원할 수 있는 것보다 많은 기능을 제공하는 경우. 예를 들어, Stage에서 마스크를 가져오거나 설정하려고 할 경우 (`stage.mask` 사용) Flash Player에서는 "The Stage class does not implement this property or method."라는 메시지와 함께 IllegalOperationError가 발생합니다. • 하위 클래스에서 필요하지 않으며 지원하기를 원치 않는 메서드를 상속하는 경우 • 액세스 가능성 기능 지원 없이 Flash Player가 컴파일되었을 때 특정 액세스 가능성 메서드가 호출되는 경우 • Flash Player의 런타임 버전에서 제작 전용 기능이 호출된 경우 • 타임라인에 있는 객체의 이름을 설정하려고 한 경우
IOError	IOError 예외는 일부 I/O 예외 유형이 발생할 경우 발생합니다.	예를 들어, 연결되지 않았거나 연결이 해제된 소켓에서 읽기/쓰기 작업을 시도한 경우 IOError 예외가 발생합니다.

클래스 이름	설명	참고
MemoryError	MemoryError 예외는 메모리 할당 요청이 실패하면 발생합니다.	기본적으로 ActionScript Virtual Machine 2는 ActionScript 프로그램에서 할당할 수 있는 메모리 크기에 제한을 두지 않습니다. 데스크톱 PC에서는 메모리 할당에 실패하는 경우가 드물지만, 시스템에서 작업에 필요한 메모리를 할당할 수 없을 경우 이 오류가 발생합니다. 그러므로 데스크톱 PC에서 이 예외는 극도로 큰 메모리 할당을 요청하는 경우가 아니면 발생하지 않습니다. 예를 들어, 32비트 Microsoft® Windows® 프로그램의 경우 액세스할 수 있는 주소 공간이 2GB로 제한되므로 30억 바이트 요청은 불가능합니다.
ScriptTimeoutError	ScriptTimeoutError 예외는 15초 스크립트 타임아웃 간격에 도달할 경우 발생합니다. ScriptTimeoutError 예외를 포착하면 스크립트 타임아웃을 보다 적절하게 처리할 수 있습니다. 예외 핸들러가 없는 경우 포착되지 않는 예외 핸들러에서 오류 메시지와 함께 대화 상자를 표시합니다.	악의적 개발자가 예외를 포착하여 무한 루프를 유지하는 것을 방지하기 위해, 특정 스크립트 동안 발생한 첫 번째 ScriptTimeoutError 예외만 포착할 수 있습니다. 이후의 ScriptTimeoutError 예외는 코드에서 포착할 수 없으며 포착되지 않는 예외 핸들러로 바로 전달됩니다.
StackOverflowError	StackOverflowError 예외는 스크립트에 사용할 수 있는 스택이 소진된 경우 발생합니다.	StackOverflowError 예외는 무한 재귀가 발생했음을 나타낼 수 있습니다.

예제: CustomErrors 응용 프로그램

CustomErrors 응용 프로그램은 응용 프로그램 작성 시 사용자 정의 오류 작업에 관련된 기술에 대해 설명합니다. 이러한 기술은 다음과 같습니다.

- XML 패킷 유효성 검사
- 사용자 정의 오류 작성
- 사용자 정의 오류 발생
- 오류 발생 시 사용자에게 알림

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. CustomErrors 응용 프로그램 파일은 Samples/CustomError 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
CustomErrors.mxml 또는 CustomErrors.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/errors/ApplicationError.as	FatalError 및 WarningError 클래스 모두에 대해 기본 오류 클래스 역할을 하는 클래스입니다.
com/example/programmingas3/errors/FatalError.as	응용 프로그램에서 발생할 수 있는 FatalError 오류를 정의하는 클래스로, 사용자 정의 ApplicationError 클래스를 확장합니다.
com/example/programmingas3/errors/Validator.as	사용자가 제공한 직원의 XML 패킷 유효성을 검사하는 단일 메서드를 정의하는 클래스입니다.
com/example/programmingas3/errors/WarningError.as	응용 프로그램에서 발생할 수 있는 WarningError 오류를 정의하는 클래스로, 사용자 정의 ApplicationError 클래스를 확장합니다.

CustomErrors 응용 프로그램 개요

CustomErrors.mxml 파일에는 사용자 정의 오류 응용 프로그램에 대한 사용자 인터페이스 및 일부 논리가 포함되어 있습니다. 응용 프로그램의 `creationComplete` 이벤트가 전달되면 `initApp()` 메서드가 호출됩니다. 이 메서드는 Validator 클래스에서 검증할 샘플 XML 패킷을 정의합니다. 다음 코드는 `initApp()` 메서드를 보여 줍니다.

```
private function initApp():void
{
  employeeXML =
    <employee id="12345">
      <firstName>John</firstName>
      <lastName>Doe</lastName>
```

```
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

XML 패킷은 Stage의 TextArea 구성 요소 인스턴스에서 나중에 표시됩니다. 그러므로 유효성을 다시 검사하기 전에 XML 패킷을 수정할 수 있습니다.

사용자가 Validate 버튼을 클릭하면 `validateData()` 메서드가 호출됩니다. 이 메서드는 Validator 클래스의 `validateEmployeeXML()` 메서드를 사용하여 직원 XML 패킷의 유효성을 검사합니다. 다음 코드는 `validateData()` 메서드를 보여 줍니다.

```
public function validateData():void
{
  try
  {
    var tempXML:XML = XML(xmlText.text);
    Validator.validateEmployeeXML(tempXML);
    status.text = "The XML was successfully validated.";
  }
  catch (error:FatalError)
  {
    showFatalError(error);
  }
  catch (error:WarningError)
  {
    showWarningError(error);
  }
  catch (error:Error)
  {
    showGenericError(error);
  }
}
```

먼저 TextArea 구성 요소 인스턴스 `xmlText`의 내용을 사용하여 임시 XML 객체가 만들어집니다. 그 다음 사용자 정의 Validator 클래스(com.example.programmingas3/errors/Validator.as)의 `validateEmployeeXML()` 메서드가 호출되고 임시 XML 객체가 매개 변수로 전달됩니다. XML 패킷이 유효한 경우 `status` Label 구성 요소 인스턴스에 성공 메시지가 표시되며 응용 프로그램이 종료됩니다. `validateEmployeeXML()` 메서드에서 사용자 정의 오류(FatalError, WarningError 또는 일반 오류 발생)가 발생한 경우 적합한 `catch` 문이 실행되며 `showFatalError()`, `showWarningError()` 또는 `showGenericError()` 메서드가 호출됩니다. 이 메서드는 각각 사용자에게 특정 오류 발생을 알리기 위해 Alert 구성 요소에 적합한 메시지를 표시하며, 특정 메시지로 `status` Label 구성 요소도 업데이트합니다.

직원 XML 패킷에 대한 유효성 검사 중 치명적 오류가 발생하면 다음 코드와 같이 Alert 구성 요소에 오류 메시지가 표시되고 `xmlText` TextArea 구성 요소 인스턴스 및 `validateBtn` Button 구성 요소 인스턴스가 비활성화됩니다.

```
public function showFatalError(error:FatalError):void
```

```
{
  var message:String = error.message + "\n\n" + "Click OK to end.";
  var title:String = error.getTitle();
  Alert.show(message, title);
  status.text = "This application has ended.";
  this.xmlText.enabled = false;
  this.validateBtn.enabled = false;
}
```

치명적 오류가 아닌 경고 오류가 발생한 경우에는 Alert 구성 요소 인스턴스에 오류 메시지가 표시되지만 TextField 및 Button 구성 요소 인스턴스는 비활성화되지 않습니다.

`showWarningError()` 메서드는 Alert 구성 요소 인스턴스에 사용자 정의 오류 메시지를 표시하며, 이 메시지는 사용자에게 XML 유효성 검증을 계속할지 아니면 스크립트를 중단할지 선택하도록 요청합니다. 다음 예제는 `showWarningError()` 메서드를 보여 줍니다.

```
public function showWarningError(error:WarningError):void
{
  var message:String = error.message + "\n\n" + "Do you want to exit this
  application?";
  var title:String = error.getTitle();
  Alert.show(message, title, Alert.YES | Alert.NO, null, closeHandler);
  status.text = message;
}
```

사용자가 Yes 또는 No 버튼을 사용하여 Alert 구성 요소 인스턴스를 닫으면 `closeHandler()` 메서드가 호출됩니다. 다음 예제는 `closeHandler()` 메서드를 보여 줍니다.

```
private function closeHandler(event:CloseEvent):void
{
  switch (event.detail)
  {
    case Alert.YES:
      showFatalError(new FatalError(9999));
      break;
    case Alert.NO:
      break;
  }
}
```

사용자가 경고 오류 Alert 대화 상자에서 Yes를 클릭하여 스크립트를 중단하도록 선택한 경우 FatalError가 발생하여 응용 프로그램이 종료됩니다.

사용자 정의 유효성 검사기 작성

사용자 정의 Validator 클래스에는 단일 메서드인 validateEmployeeXML()이 포함됩니다. validateEmployeeXML() 메서드는 유효성을 검사하려는 XML 패킷인 employee를 단일 인수로 취합니다. validateEmployeeXML() 메서드는 다음과 같습니다.

```
public static function validateEmployeeXML(employee:XML):void
{
  // XML 항목의 무결성을 검사합니다 .
  if (employee.costCenter.length() < 1)
  {
    throw new FatalError(9000);
  }
  if (employee.costCenter.length() > 1)
  {
    throw new WarningError(9001);
  }
  if (employee.ssn.length() != 1)
  {
    throw new FatalError(9002);
  }
}
```

유효성을 검사하기 위해서는 직원이 단일 비용 센터에 속해 있어야 합니다. 직원이 어느 비용 센터에도 속해 있지 않은 경우 메서드에서 FatalError가 발생하며, 이는 기본 응용 프로그램 파일에서 validateData() 메서드가 실행되도록 합니다. 직원이 둘 이상의 비용 센터에 속해 있는 경우 WarningError가 발생합니다. XML 유효성 검사기의 최종 점검 절차는 사용자의 주민등록 번호가 정확하게 정의되었는지 확인하는 것입니다(XML 패킷의 ssn 노드). 정확히 하나의 ssn 노드가 없는 경우 FatalError 오류가 발생합니다.

validateEmployeeXML() 메서드에 ssn 노드에 유효한 번호가 포함되었는지 또는 직원에게 최소한 하나의 전화 번호 및 전자 메일 주소가 정의되어 있는지 및 두 값이 모두 유효한지 등의 점검 사항을 추가할 수 있습니다. 또한 XML을 수정하여 각 직원이 고유한 ID를 가지고 해당 관리자 ID를 지정하도록 할 수 있습니다.

ApplicationError 클래스 정의

ApplicationError 클래스는 FatalError 및 WarningError 클래스 모두에 대해 기본 클래스 역할을 합니다. ApplicationError 클래스는 Error 클래스를 확장하며 오류 ID, 심각도 및 사용자 정의 오류 코드 및 메시지를 포함하는 XML 객체를 비롯한 고유한 사용자 정의 메서드 및 속성을 정의합니다. 이 클래스는 또한 각 오류 유형의 심각도를 정의하는 데 사용되는 두 개의 정적 상수도 정의합니다.

ApplicationError 클래스 생성자 메서드는 다음과 같습니다.

```
public function ApplicationError()
{
```

```
  messages =
    <errors>
      <error code="9000">
        <![CDATA[Employee must be assigned to a cost center.]]>
      </error>
      <error code="9001">
        <![CDATA[Employee must be assigned to only one cost center.]]>
      </error>
      <error code="9002">
        <![CDATA[Employee must have one and only one SSN.]]>
      </error>
      <error code="9999">
        <![CDATA[The application has been stopped.]]>
      </error>
    </errors>;
}
```

XML 객체의 각 오류 노드에는 고유한 숫자 코드 및 오류 메시지가 포함됩니다. 오류 메시지는 다음 `getMessageText()` 메서드에서 보는 바와 같이 E4X를 사용하여 오류 코드별로 쉽게 조회할 수 있습니다.

```
public function getMessageText(id:int):String
{
  var message:XMLList = messages.error.(@code == id);
  return message[0].text();
}
```

`getMessageText()` 메서드는 단일 정수 인수인 `id`를 취하여, 문자열을 반환합니다. `id` 인수는 조회할 오류에 대한 오류 코드입니다. 예를 들어, `id`로 9001을 전달하면 직원은 하나의 비용 센터에만 할당되어야 한다는 오류 메시지가 검색됩니다. 둘 이상의 오류에 동일한 오류 코드가 있을 경우 ActionScript는 발견된 첫 번째 결과에 대해서만 오류 메시지를 반환합니다(반환된 XMLList 객체의 `message[0]`).

이 클래스의 다음 메서드인 `getTitle()`은 아무 매개 변수도 취하지 않으며, 이 특정 오류의 오류 ID가 포함된 문자열 값을 반환합니다. 이 값은 Alert 구성 요소의 제목에 사용되어 XML 패킷 유효성 검사 중 발생한 오류를 쉽고 정확하게 식별하는 데 도움을 줍니다. 다음 예제는 `getTitle()` 메서드를 보여 줍니다.

```
public function getTitle():String
{
  return "Error #" + id;
}
```

ApplicationError 클래스의 마지막 메서드는 `toString()`입니다. 이 메서드는 Error 클래스에서 정의된 함수를 대체하므로, 이를 통해 오류 메시지의 표시 형식을 사용자 정의할 수 있습니다. 이 메서드는 발생한 특정 오류 번호 및 메시지를 식별하는 문자열을 반환합니다.

```
public override function toString():String
{
  return "[APPLICATION ERROR #" + id + "] " + message;
}
```

FatalError 클래스 정의

FatalError 클래스는 사용자 정의 ApplicationError 클래스를 확장하며, FatalError 생성자, `getTitle()` 및 `toString()`의 세 가지 메서드를 정의합니다. 첫 번째 메서드인 FatalError 생성자는 단일 정수 인수인 `errorID`를 취하며, ApplicationError 클래스에서 정의된 정적 상수 값을 사용하여 오류 심각도를 설정하고, ApplicationError 클래스의 `getMessageText()` 메서드를 호출하여 특정 오류의 오류 메시지를 가져옵니다. FatalError 생성자는 다음과 같습니다.

```
public function FatalError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.FATAL;
  message = getMessageText(errorID);
}
```

FatalError 클래스의 다음 메서드인 `getTitle()`은 ApplicationError 클래스에서 이전에 정의된 `getTitle()` 메서드를 대체하며, 제목에 "-- FATAL" 텍스트를 추가하여 사용자에게 치명적 오류가 발생했음을 알립니다. `getTitle()` 메서드는 다음과 같습니다.

```
public override function getTitle():String
{
  return "Error #" + id + " -- FATAL";
}
```

이 클래스의 마지막 메서드인 `toString()`은 ApplicationError 클래스에서 정의된 `toString()` 메서드를 대체합니다. `toString()` 메서드는 다음과 같습니다.

```
public override function toString():String
{
  return "[FATAL ERROR #" + id + "] " + message;
}
```

WarningError 클래스 정의

WarningError 클래스는 ApplicationError 클래스를 확장하며 다음 코드와 같이 문자열이 조금 변경되고 오류 심각도를 ApplicationError.FATAL이 아닌 ApplicationError.WARNING으로 설정하는 것만 제외하면 FatalError 클래스와 거의 동일합니다.

```
public function WarningError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.WARNING;
  message = super.getMessageText(errorID);
}
```

제 9 장

9 일반 표현식 사용

일반 표현식은 문자열에서 일치하는 텍스트를 찾고 조작하는 데 사용되는 패턴을 기술합니다. 이러한 일반 표현식은 문자열과 비슷하지만 패턴과 반복을 기술하는 특수 코드를 포함할 수 있습니다. 예를 들어, 다음 일반 표현식은 A로 시작하고 그 뒤에 하나 이상의 숫자가 순차적으로 나오는 문자열을 찾습니다.

```
/A\d+/
```

이 장에서는 일반 표현식을 구성하는 기본 구문에 대해 설명합니다. 그러나 일반 표현식은 복잡하고 함축적일 수 있으므로 웹 또는 서점에서 일반 표현식에 대한 자세한 리소스를 참조하면 도움이 됩니다. 또한 프로그래밍 환경에 따라 다양한 방법으로 일반 표현식을 구현한다는 사실을 기억해야 합니다. ActionScript 3.0에서는 ECMAScript 버전 3 언어 사양(ECMA-262)에 정의된 일반 표현식을 구현합니다.

목차

일반 표현식의 기초

일반 표현식 사용 소개

일반 표현식은 문자 패턴을 기술합니다. 일반 표현식은 사용자가 입력한 전화 번호의 자릿수가 올바른지 확인하는 것처럼 텍스트 값이 특정 패턴을 따르는지 확인하거나, 텍스트 값에서 특정 패턴과 일치하는 부분을 바꾸는 데 주로 사용됩니다.

일반 표현식은 단순하게 나타낼 수 있습니다. 예를 들어, 특정 문자열이 "ABC"와 일치하는지 확인하거나 문자열에 있는 모든 "ABC"를 다른 텍스트로 대체하려 할 경우, 다음과 같은 일반 표현식을 사용하여 A, B, C 문자가 차례로 포함된 패턴을 정의할 수 있습니다.

```
/ABC/
```

일반 표현식 리터럴은 슬래시(/) 문자로 나타냅니다.

일반 표현식 패턴은 유효한 전자 메일 주소를 찾는 다음 표현식과 같이 복잡할 수도 있으며 경우에 따라 암호처럼 보일 수도 있습니다.

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

일반 표현식은 문자열의 패턴을 찾아서 문자를 바꿀 때 흔히 사용됩니다. 이와 같은 경우 일반 표현식 객체를 만들어 여러 String 클래스 메서드 중 하나에 대한 매개 변수로 사용할 수 있습니다. String 클래스의 `match()`, `replace()`, `search()` 및 `split()` 메서드는 일반 표현식을 매개 변수로 사용합니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 "문자열의 패턴 찾기 및 하위 문자열 바꾸기"를 참조하십시오.

RegExp 클래스에는 `test()` 및 `exec()` 메서드가 포함되어 있습니다. 자세한 내용은 287페이지의 "문자열에 일반 표현식을 사용하는 데 필요한 메서드"를 참조하십시오.

일반적인 일반 표현식 작업

일반 표현식은 이 장에서 자세히 설명하고 있는 것과 같이 여러 다양한 용도로 사용할 수 있습니다.

- 일반 표현식 패턴 만들기
- 패턴에서 특수 문자 사용
- 여러 문자의 시퀀스 식별(예: "2자리 숫자" 또는 "7개 문자와 10개 문자 사이")
- 문자 또는 숫자 범위 내 문자 식별(예: "*a*에서부터 *m* 사이의 임의의 문자")
- 가능한 문자 세트에서 문자 식별
- 하위 시퀀스 식별(패턴 내 세그먼트)
- 패턴을 사용한 텍스트 비교 및 바꾸기

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- 이스케이프 문자: 다음에 오는 문자가 리터럴 문자가 아닌 메타문자로 처리되어야 함을 나타내는 문자입니다. 일반 표현식 구문에서 백슬래시 문자(\)는 이스케이프 문자로, 백슬래시 뒤에 오는 문자는 일반 문자가 아닌 특수 코드로 처리됩니다.
- 플래그: 일반 표현식 패턴 사용 방법에 대한 옵션을 지정하는 문자입니다(예: 대/소문자 구분 여부).
- 메타문자: 일반 표현식 패턴에서 특별한 의미를 갖는 문자로, 해당 패턴에서 문자를 글자 그대로 나타내는 문자와 대비되는 개념입니다.
- 한정 기호: 패턴을 구성하는 한 부분의 반복 횟수를 나타내는 문자(또는 여러 문자)입니다. 예를 들어, 우편 번호를 반드시 5자리 또는 9자리 숫자로 구성하도록 지정할 때 한정 기호를 사용할 수 있습니다.
- 일반 표현식: 문자 패턴을 정의하는 프로그램 명령으로, 해당 패턴에 일치하는 패턴을 찾거나 문자열의 일부를 바꿀 때 사용할 수 있습니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플은 주로 일반 표현식 패턴으로 구성되어 있으므로 예제를 테스트하려면 다음 단계를 수행합니다.

1. 새 Flash 문서를 만듭니다.
2. 키프레임을 선택하고 [액션] 패널을 엽니다.
3. 다음과 같이 RegExp(일반 표현식) 변수를 만듭니다.

```
var pattern:RegExp = /ABC/;
```

4. 예제에서 패턴을 복사하고 RegExp 변수의 값으로 할당합니다. 예를 들어, 바로 앞의 코드 행에서 패턴은 세미콜론을 제외한 코드의 등호 오른쪽 부분입니다(/ABC/).
5. 일반 표현식을 테스트하기 위한 해당 문자열을 포함하여 한 개 이상의 String 변수를 만듭니다. 예를 들어, 유효한 전자 메일 주소를 테스트하는 일반 표현식을 만드는 경우 유효한 전자 메일 주소와 잘못된 전자 메일 주소를 포함하여 여러 개의 String 변수를 만듭니다.

```
var goodEmail:String = "bob@example.com";
var badEmail:String = "5@$2.99";
```

6. String 변수를 테스트할 코드 행을 추가하여 해당 변수가 일반 표현식 패턴과 일치하는지 여부를 확인합니다. trace() 함수를 사용하거나 스테이지의 텍스트 필드에 기록함으로써 화면에 출력할 값은 패턴과 일치하는 변수입니다.

```
trace(goodEmail, " is valid:", pattern.test(goodEmail));
trace(badEmail, " is valid:", pattern.test(badEmail));
```

예를 들어, pattern이 유효한 전자 메일 주소에 대한 일반 표현식 패턴을 정의할 경우 앞의 코드 행은 이 텍스트를 [출력] 패널에 기록합니다.

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

스테이지의 텍스트 필드 인스턴스에 값을 기록하거나 trace() 함수를 사용하여 [출력] 패널에 값을 인쇄함으로써 값을 테스트하는 방법에 대한 자세한 내용은 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"를 참조하십시오.

일반 표현식 구문

이 단원에서는 ActionScript 일반 표현식 구문의 모든 요소에 대해 설명합니다. 일반 표현식은 복잡하고 함축적일 수 있으므로 웹 또는 서점에서 일반 표현식에 대한 자세한 리소스를 참조하면 도움이 됩니다. 또한 프로그래밍 환경에 따라 다양한 방법으로 일반 표현식을 구현한다는 사실을 기억해야 합니다. ActionScript 3.0에서는 ECMAScript 버전 3 언어 사양(ECMA-262)에 정의된 일반 표현식을 구현합니다.

일반적으로 간단한 문자열보다 복잡한 패턴과 일치하는 일반 표현식을 사용합니다. 예를 들어, 다음 일반 표현식은 A, B, C 문자가 차례로 나오고 그 다음에 숫자가 나오는 패턴을 정의합니다.

```
/ABC\d/
```

여기에서 \d 코드는 "임의의 숫자"를 나타냅니다. 백슬래시(\) 문자는 이스케이프 문자라고 하며 뒤에 나오는 문자(이 예제에서는 d)와 결합할 경우 일반 표현식에서 특별한 의미를 갖습니다. 이 장에서는 이러한 이스케이프 문자 시퀀스 및 기타 일반 표현식 구문 기능에 대해 설명합니다.

다음 일반 표현식은 ABC 문자 다음에 숫자(개수 제한 없음)가 나오는 패턴을 정의합니다. 이때 별표에 주의하십시오.

```
/ABC\d*/
```

별표 문자(*)는 *메타문자*입니다. 메타문자는 일반 표현식에서 특별한 의미를 갖는 문자입니다. 별표는 메타문자의 특정 유형으로 *한정 기호*라고 하며 반복되는 단일 문자 또는 문자 그룹의 범위(개수)를 지정하는 데 사용됩니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.

일반 표현식에는 패턴뿐만 아니라 플래그를 포함하여 일반 표현식을 일치시키는 방식을 지정할 수 있습니다. 예를 들어, 다음 일반 표현식에서는 i 플래그를 사용하여 일치하는 문자열에서 대/소문자를 구분하지 않도록 지정합니다.

```
/ABC\d*/i
```

자세한 내용은 283페이지의 "플래그 및 속성"을 참조하십시오.

String 클래스의 `match()`, `replace()` 및 `search()` 메서드와 함께 일반 표현식을 사용할 수 있습니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 "문자열의 패턴 찾기 및 하위 문자열 바꾸기"를 참조하십시오.

일반 표현식 인스턴스 만들기

두 가지 방법을 사용하여 일반 표현식 인스턴스를 만들 수 있는데, 첫 번째 방법은 슬래시 문자(/)를 사용하여 일반 표현식을 나타내고 다른 방법은 `new` 생성자를 사용합니다. 예를 들어, 다음 일반 표현식은 동일합니다.

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

슬래시는 따옴표를 사용하여 문자열 리터럴을 나타내는 것과 같은 방식으로 일반 표현식 리터럴을 나타냅니다. 슬래시 내의 일반 표현식 부분은 *패턴*을 정의합니다. 또한 일반 표현식에서 마지막 슬래시 뒤에 *플래그*를 포함할 수도 있습니다. 이러한 플래그는 일반 표현식의 일부로 간주되지만 패턴과 별개입니다.

`new` 생성자를 사용할 때는 두 문자열을 사용하여 일반 표현식을 정의하는데, 다음 예제와 같이 첫 번째 문자열은 패턴을 정의하고 두 번째 문자열은 플래그를 정의합니다.

```
var pattern2:RegExp = new RegExp("bob", "i");
```

슬래시 기호를 사용하여 정의된 일반 표현식 *내에* 다시 슬래시를 포함하는 경우에는 슬래시 앞에 백슬래시(\) 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식은 `1/2` 패턴과 일치하는 문자열을 찾습니다.

```
var pattern:RegExp = /1\/2/;
```

`new` 생성자를 사용하여 정의된 일반 표현식 *내에* 따옴표를 포함하려면 문자열 리터럴을 정의할 때처럼 따옴표 앞에 백슬래시(\) 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식은 `eat at "joe's"` 패턴과 일치하는 문자열을 찾습니다.

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

슬래시 기호를 사용하여 정의된 일반 표현식에서 백슬래시 이스케이프 문자를 따옴표와 함께 사용하면 안 됩니다. 마찬가지로 `new` 생성자를 사용하여 정의된 일반 표현식에서 이스케이프 문자를 슬래시와 함께 사용하면 안 됩니다. 다음 일반 표현식은 동일하며 `1/2 "joe's"` 패턴을 정의합니다.

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

또한 `new` 생성자를 사용하여 정의된 일반 표현식에서 임의의 숫자와 일치하는 `\d`와 같이 백슬래시(\) 문자로 시작하는 메타시퀀스를 사용하려면 백슬래시 문자를 두 번 입력합니다.

```
var pattern:RegExp = new RegExp("\\d+", ""); // 하나 이상의 숫자를 찾습니다 .
```

이 경우 RegExp() 생성자 메서드의 첫 번째 매개 변수가 문자열이고 문자열 리터럴에서 단일 백슬래시 문자를 인식하려면 백슬래시 문자를 두 번 입력해야 하므로 결과적으로 백슬래시 문자를 두 번 입력해야 합니다.

다음에 나오는 단원에서는 일반 표현식 패턴을 정의하는 구문에 대해 설명합니다.

플래그에 대한 자세한 내용은 283페이지의 "플래그 및 속성"을 참조하십시오.

문자, 메타문자 및 메타시퀀스

가장 간단한 일반 표현식은 다음 예제와 같이 문자 시퀀스와 일치하는 일반 표현식입니다.

```
var pattern:RegExp = /hello/;
```

그러나 메타문자라고 하는 다음 문자는 일반 표현식에서 특별한 의미를 갖습니다.

```
^ $ \ . * + ? ( ) [ ] { } |
```

예를 들어, 다음 일반 표현식은 A 문자 다음에 B 문자 인스턴스가 0개 이상 나오고(별표 메타문자는 이 반복을 나타냄) 그 다음에 C 문자가 나오는 문자열을 찾습니다.

```
/AB*C/
```

일반 표현식 패턴에서 특별한 의미가 없는 메타문자를 포함하려면 백슬래시(\) 이스케이프 문자를 사용해야 합니다. 예를 들어, 다음 일반 표현식은 A 문자, B 문자, 별표, C 문자가 차례로 나오는 문자열을 찾습니다.

```
var pattern:RegExp = /AB\*C/;
```

메타문자와 같이 *메타시퀀스*도 일반 표현식에서 특별한 의미를 갖습니다. 메타시퀀스는 둘 이상의 문자로 구성됩니다. 다음 단원에서는 메타문자와 메타시퀀스를 사용하는 방법에 대해 자세히 설명합니다.

메타문자 정보

다음 표에는 일반 표현식에 사용할 수 있는 메타문자가 요약되어 있습니다.

메타문자	설명
^(캐럿)	문자열의 시작 부분에서 찾습니다. m(multiline) 플래그를 설정하면 캐럿 문자는 행의 시작 부분도 찾습니다(285페이지의 "m(multiline) 플래그" 참조). 또한 문자 클래스의 맨 처음에 사용되는 경우에는 문자열의 시작 지점이 아니라 부정을 나타냅니다. 자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.
$(달러 기호)	문자열의 끝 부분에서 찾습니다. m(multiline) 플래그를 설정하면 $는 개행(\n) 문자 앞의 위치도 찾습니다. 자세한 내용은 285페이지의 "m(multiline) 플래그"를 참조하십시오.

메타문자	설명
\(백슬래시)	특수 문자의 특별한 메타문자 의미를 이스케이프합니다. 또한 `/1\/2/`와 같이 문자 1, 슬래시 문자, 문자 2가 차례로 나오도록 슬래시 문자를 일반 표현식 리터럴로 사용하려는 경우에도 백슬래시 문자를 사용합니다.
.(도트)	임의의 단일 문자를 찾습니다. 도트는 s(`dotall`) 플래그가 설정된 경우에만 개행 문자(`\n`)를 찾습니다. 자세한 내용은 285페이지의 "s(dotall) 플래그"를 참조하십시오.
*(별표)	바로 앞의 항목이 0번 이상 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
+(더하기)	바로 앞의 항목이 1번 이상 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
?(물음표)	바로 앞의 항목이 0번 또는 1번 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
`(, )`	일반 표현식 내에서 그룹을 정의합니다. 다음과 같은 경우 그룹을 사용합니다. • | 범위를 제한하려는 경우(예: `/(a|b|c)d/`) • 한정 기호 범위를 정의하려는 경우(예: `/(walla.){1,2}/`) • 역참조를 사용하는 경우. 예를 들어, 다음 일반 표현식에서 `\1`은 패턴의 첫 번째 괄호 그룹과 일치하는 모든 항목을 찾습니다. `/(\w*) is repeated: \1/` 자세한 내용은 280페이지의 "그룹"을 참조하십시오.
`[, ]`	단일 문자에 대해 일치하는 항목을 정의하는 문자 클래스를 정의합니다. `/[aeiou]/` 지정된 문자 중 하나를 찾습니다. 문자 클래스 내에서 문자 범위를 지정하려면 하이픈(-)을 사용합니다. `/[A-Z0-9]/` 대문자 A~Z 또는 0~9를 찾습니다. 문자 클래스 내에서] 및 - 문자를 이스케이프하려면 백슬래시를 삽입합니다. `/[+\-]\d+/` 하나 이상의 숫자 앞에서 + 또는 -를 찾습니다. 문자 클래스 내에서 다른 문자(일반적으로 메타문자)는 메타문자가 아니라 일반 문자로 처리되므로 백슬래시를 사용할 필요가 없습니다. `/[$£]/` $ 또는 £ 문자를 찾습니다. 자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.
|(*파이프*)	왼쪽 항목 또는 오른쪽 항목 중 하나를 찾습니다. `/abc|xyz/` abc 또는 xyz를 찾습니다.

메타시퀀스 정보

메타시퀀스는 일반 표현식 패턴에서 특별한 의미를 갖는 문자 시퀀스입니다. 다음 표에서는 이러한 메타시퀀스에 대해 설명합니다.

메타시퀀스	설명
{*n*} {*n*,} 및 {*n*,*n*}	앞의 항목에 대한 숫자 한정 기호 또는 한정 기호 범위를 지정합니다. /A{27}/ 27번 반복된 A 문자를 찾습니다. /A{3,}/ 3번 이상 반복된 A 문자를 찾습니다. /A{3,5}/ 3~5번 반복된 A 문자를 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
\b	단어 문자와 단어가 아닌 문자 사이의 위치에서 찾습니다. 또한 문자열의 첫 번째 문자 또는 마지막 문자가 단어이면 문자열의 시작 또는 끝 부분도 찾습니다.
\B	두 개의 단어 문자 사이의 위치에서 찾습니다. 또한 단어가 아닌 두 문자 사이의 위치에서도 찾습니다.
\d	10진수를 찾습니다.
\D	숫자 이외의 다른 문자를 찾습니다.
\f	용지 공급 문자를 찾습니다.
\n	개행 문자를 찾습니다.
\r	캐리지 리턴 문자를 찾습니다.
\s	빈 칸, 탭, 개행 문자 또는 캐리지 리턴 문자 등을 포함하여 모든 공백 문자를 찾습니다.
\S	공백이 아닌 다른 문자를 찾습니다.
\t	탭 문자를 찾습니다.
\u*nnnn*	16진수 *nnnn*으로 지정된 문자 코드를 사용하는 유니코드 문자를 찾습니다. 예를 들어, \u263a는 스마일 문자입니다.
\v	수직 탭 문자를 찾습니다.
\w	단어 문자(A-Z, a-z, 0-9 또는 _)를 찾습니다. 이때 \w는 é, ñ 또는 ç 등 영어 이외의 문자는 찾지 않습니다.
\W	단어가 아닌 다른 문자를 찾습니다.
\x*nn*	16진수 *nn*으로 정의된 ASCII 값을 사용하는 문자를 찾습니다.

문자 클래스

문자 클래스를 사용하면 일반 표현식에서 한 위치에 대응되는 문자 목록을 지정할 수 있습니다. 문자 클래스를 정의할 때는 대괄호(`[ ]`)를 사용합니다. 예를 들어, 다음 일반 표현식은 `bag`, `beg`, `big`, `bog` 또는 `bug`와 일치하는 문자 클래스를 정의합니다.

```
/b[aeiou]g/
```

문자 클래스의 이스케이프 시퀀스

대개 일반 표현식에서 특별한 의미를 갖는 메타문자와 메타시퀀스의 대부분은 문자 클래스 내에서는 같은 의미를 갖지 *않습니다*. 예를 들어, 별표는 일반 표현식에서 반복을 나타내는 데 사용되지만 문자 클래스에 나타날 때는 그렇지 않습니다. 다음 문자 클래스는 나열된 다른 모든 문자와 함께 별표를 문자 그대로 찾습니다.

```
/[abc*123]/
```

그러나 다음 표에 나열된 세 단어는 메타문자로 사용되어 문자 클래스에서도 특별한 의미를 갖습니다.

메타문자	문자 클래스에 사용할 경우 의미
`]`	문자 클래스의 끝을 정의합니다.
`-`	문자 범위를 정의합니다(278페이지의 “문자 클래스의 문자 범위” 참조).
`\`	메타시퀀스를 정의하고 메타문자의 특별한 의미를 제거합니다.

이러한 문자가 특별한 메타문자 의미를 갖지 않고 리터럴 문자로 인식되게 하려면 해당 문자 앞에 백슬래시 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식에는 네 가지 심볼(`$`, `\`, `]` 또는 `-`) 중 하나와 일치하는 문자 클래스가 포함되어 있습니다.

```
/[$\\\]\-]/
```

특별한 의미를 유지하는 메타문자뿐만 아니라 다음 메타시퀀스도 문자 클래스 내에서 특별한 의미를 갖는 메타시퀀스로 사용됩니다.

메타시퀀스	문자 클래스에 사용할 경우 의미
`\n`	개행 문자를 찾습니다.
`\r`	캐리지 리턴 문자를 찾습니다.
`\t`	탭 문자를 찾습니다.
`\u`*nnnn*	16진수 *nnnn*으로 정의된 유니코드 값을 사용하는 문자를 찾습니다.
`\x`*nn*	16진수 *nn*으로 정의된 ASCII 값을 사용하는 문자를 찾습니다.

다른 일반 표현식 메타시퀀스 및 메타문자는 문자 클래스 내에서 일반 문자로 처리됩니다.

문자 클래스의 문자 범위

하이픈을 사용하면 A-Z, a-z 또는 0-9와 같이 문자 범위를 지정할 수 있습니다. 이러한 문자는 문자 세트에서 유효한 범위를 구성해야 합니다. 예를 들어, 다음 문자 클래스는 a-z 범위의 문자 중 하나 또는 임의의 숫자를 찾습니다.

```
/[a-z0-9]/
```

\x*nn* ASCII 문자 코드를 사용하여 ASCII 값으로 범위를 지정할 수도 있습니다. 예를 들어, 다음 문자 클래스는 확장 ASCII 문자(예: é, ê) 세트의 문자를 찾습니다.

```
/[\x80-\x9A]/
```

문자 클래스 부정

캐럿(^) 문자를 문자 클래스의 맨 앞에 사용하면 해당 클래스를 제외하고 나열되지 않은 모든 문자를 찾습니다. 다음 문자 클래스는 소문자(a–z) 또는 숫자 *이외의* 모든 문자를 찾습니다.

```
/[^a-z0-9]/
```

부정을 나타내려면 캐럿(^) 문자를 문자 클래스의 *맨 앞에* 입력해야 합니다. 그렇지 않으면 캐럿 문자가 문자 클래스의 문자에 추가됩니다. 예를 들어, 다음 문자 클래스는 캐럿을 포함하여 많은 심볼 문자 중 하나를 찾습니다.

```
/[!.,#+*%$&^]/
```

한정 기호

한정 기호를 사용하면 다음과 같이 패턴에서 문자 또는 시퀀스의 반복을 지정할 수 있습니다.

한정 기호 메타문자	설명
*(별표)	바로 앞의 항목이 0번 이상 반복된 것을 찾습니다.
+(더하기)	바로 앞의 항목이 1번 이상 반복된 것을 찾습니다.
?(물음표)	바로 앞의 항목이 0번 또는 1번 반복된 것을 찾습니다.
{*n*} {*n*,} 및 {*n*,*n*}	앞의 항목에 대한 숫자 한정 기호 또는 한정 기호 범위를 지정합니다. /A{27}/ 27번 반복된 A 문자를 찾습니다. /A{3,}/ 3번 이상 반복된 A 문자를 찾습니다. /A{3,5}/ 3-5번 반복된 A 문자를 찾습니다.

한정 기호를 단일 문자, 문자 클래스 또는 그룹에 적용할 수 있습니다.

- /a+/ 1번 이상 반복된 a 문자를 찾습니다.
- /\d+/ 하나 이상의 숫자를 찾습니다.
- /[abc]+/ a, b 또는 c 중에서 하나 이상의 문자가 반복되는 항목을 찾습니다.
- /(very,)*/ very라는 단어와 쉼표, 공백이 차례로 0번 이상 반복되는 항목을 찾습니다.

한정 기호가 적용된 괄호 그룹 내에서 한정 기호를 사용할 수 있습니다. 예를 들어, 다음 한정 기호는 `word` 및 `word-word-word`와 같은 문자열을 찾습니다.

```
/\w+(-\w+)*/
```

기본적으로 일반 표현식은 *최장 일치*를 수행합니다. 즉, 일반 표현식의 하위 패턴(예: `.*`)은 문자열에서 가능한 한 많은 문자를 찾은 후 해당 일반 표현식의 다음 부분으로 이동합니다. 예를 들어, 다음 일반 표현식과 문자열을 검토해 보십시오.

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

이 일반 표현식은 전체 문자열을 찾습니다.

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

하지만 `<p>...</p>` 그룹을 하나만 찾으려는 경우를 가정해 봅니다. 다음과 같이 하면 이렇게 할 수 있습니다.

```
<p>Paragraph 1</p>
```

한정 기호를 *최단 일치 한정 기호*로 바꾸려면 해당 한정 기호 뒤에 물음표(`?`)를 추가합니다. 예를 들어, 최단 일치 한정 기호 `*?`를 사용하는 다음 일반 표현식은 `<p>`, 최소 문자 수, `</p>`가 차례로 나오는 항목을 찾습니다.

```
/<p>.*?<\/p>/
```

한정 기호를 사용할 경우에는 다음과 같은 내용을 항상 기억해야 합니다.

- `{0}` 및 `{0,0}` 한정 기호는 일치시킬 때 항목을 제외하지 않습니다.
- `/abc+*/`와 같이 여러 한정 기호를 함께 사용하지 마십시오.
- 도트(`.`)는 `*` 한정 기호가 뒤에 나오더라도 `s(dotall)` 플래그가 설정되어 있지 않으면 행을 확장하지 않습니다. 예를 들어, 다음과 같은 코드를 살펴봅니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
  // 출력 : <p>Test
  //          Multiline</p>
```

자세한 내용은 285페이지의 “s(dotall) 플래그”를 참조하십시오.

여러 항목 중 하나 선택

일반 표현식에 |(파이프) 문자를 사용하면 일반 표현식 엔진에서 여러 항목 중 하나를 찾습니다. 예를 들어, 다음 일반 표현식은 cat, dog, pig, rat 단어 중 하나를 찾습니다.

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

괄호를 사용하여 그룹을 정의하면 | 문자의 범위를 제한할 수 있습니다. 다음 일반 표현식은 nap 또는 nip라는 단어가 뒤에 나오는 cat을 찾습니다.

```
var pattern:RegExp = /cat(nap|nip)/;
```

자세한 내용은 280페이지의 "그룹"을 참조하십시오.

각각 | 문자와 문자 클래스([및] 문자를 사용하여 정의)를 사용하는 다음 두 일반 표현식은 동일합니다.

```
/1|3|5|7|9/
/[13579]/
```

자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.

그룹

다음과 같이 괄호를 사용하여 일반 표현식에서 그룹을 지정할 수 있습니다.

```
/class-(\d*)/
```

그룹은 패턴의 하위 영역입니다. 그룹을 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- 둘 이상의 문자에 한정 기호를 적용합니다.
- | 문자를 사용하여 선택할 하위 패턴을 기술합니다.
- 하위 문자열 일치 항목을 캡처합니다. 예를 들어, 일반 표현식에 \1을 사용하여 이전의 일치 그룹을 찾거나, 이와 비슷하게 String 클래스의 replace() 메서드에서 $1을 사용합니다.

다음 단원에서는 이러한 그룹을 사용하는 방법에 대해 자세히 설명합니다.

한정 기호와 함께 그룹 사용

그룹을 사용하지 않으면 한정 기호는 다음과 같이 해당 한정 기호 앞에 나오는 문자 또는 문자 클래스에 적용됩니다.

```
var pattern:RegExp = /ab*/ ;
// 뒤에 b 문자가 0 개 이상 나오는 a 문자를 찾습니다 .

pattern = /a\d+/;
// 뒤에 하나 이상의 숫자가 나오는 a 문자를 찾습니다 .

pattern = /a[123]{1,3}/;
// 뒤에 1, 2 또는 3 이 1-3 개 나오는 a 문자를 찾습니다 .
```

그러나 그룹을 사용하면 둘 이상의 문자 또는 문자 클래스에 한정 기호를 적용할 수 있습니다.

```
var pattern:RegExp = /(ab)*/;
// ababab 와 같이 뒤에 b 문자가 나오는 a 문자를 0 개 이상 찾습니다 .

pattern = /(a\d)+/;
// a1a5a8a3 와 같이 뒤에 숫자가 나오는 a 문자를 0 개 이상 찾습니다 .

pattern = /(spam ){1,3}/;
// 뒤에 공백이 나오는 단어 spam 을 1-3 개 찾습니다 .
```

한정 기호에 대한 자세한 내용은 278페이지의 “한정 기호”를 참조하십시오.

| 문자와 함께 그룹 사용

그룹을 사용하면 다음과 같이 | 문자를 적용할 문자 그룹을 정의할 수 있습니다.

```
var pattern:RegExp = /cat|dog/;
// cat 또는 dog 를 찾습니다 .

pattern = /ca(t|d)og/;
// catog 또는 cadog 를 찾습니다 .
```

그룹을 사용하여 하위 문자열 일치 항목 캡처

패턴에서 표준 괄호 그룹을 정의하는 경우 나중에 일반 표현식에서 이 그룹을 참조할 수 있습니다. 이를 *역참조*라고 하고 이러한 그룹을 *캡처 그룹*이라고 합니다. 예를 들어, 다음 일반 표현식에서 \1 시퀀스는 캡처 괄호 그룹과 일치하는 모든 하위 문자열을 찾습니다.

```
var pattern:RegExp = /(\d+)-by-\1/;
// 다음을 찾습니다 . 48-by-48
```

\1, \2, ... , \99를 입력하여 일반 표현식에서 이러한 역참조를 최대 99개까지 지정할 수 있습니다.

마찬가지로 String 클래스의 replace() 메서드에서 $1-$99를 사용하여, 캡처한 그룹 하위 문자열 일치 항목을 대체 문자열에 삽입할 수 있습니다.

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
// 출력 : Bob, hello.
```

또한 캡처 그룹을 사용하는 경우 RegExp 클래스의 exec() 메서드 및 String 클래스의 match() 메서드는 캡처 그룹과 일치하는 하위 문자열을 반환합니다.

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
   // bob@test.com,bob,example,com
```

비캡처 그룹 및 예측 그룹 사용

비캡처 그룹은 그룹화에만 사용되는 그룹으로, "수집"되지 않으며 번호가 지정된 역참조와 일치하지 않습니다. 비캡처 그룹을 정의하려면 다음과 같이 (?: 및)를 사용하십시오.

```
var pattern = /(?:com|org|net);
```

예를 들어, (com|org)를 각각 캡처 그룹에 추가하는 것과 비캡처 그룹에 추가하는 경우의 차이점에 주의하십시오. exec() 메서드는 완전히 일치하는 항목 다음에 캡처 그룹을 나열합니다.

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example
```

비캡처 그룹에는 *예측 그룹*이라는 특수한 유형의 그룹이 있는데, 이 그룹은 *긍정적 예측 그룹*과 *부정적 예측 그룹*이라는 두 가지 유형으로 구분됩니다.

(?= 및)를 사용하여 긍정적 예측 그룹을 정의할 수 있습니다. 이 그룹은 그룹 내의 하위 패턴이 지정된 위치에서 일치해야 함을 지정합니다. 그러나 긍정적 예측 그룹과 일치하는 문자열 부분은 일반 표현식의 나머지 패턴과도 일치할 수 있습니다. 예를 들어, 다음 코드에서 (?=e)는 긍정적 예측 그룹이므로 이 그룹과 일치하는 e는 일반 표현식의 나머지 부분(이 코드에서는 캡처 그룹 \w*))과도 일치할 수 있습니다.

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

(?! 및)를 사용하여 부정적 예측 그룹을 정의할 수 있습니다. 이 그룹은 그룹 내의 하위 패턴이 지정된 위치에서 일치하지 *않아야* 함을 지정합니다. 예를 들면 다음과 같습니다.

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

명명된 그룹 사용

명명된 그룹은 일반 표현식에서 특정 식별자가 지정된 그룹 유형입니다. 명명된 그룹을 정의하려면 (?P<name> 및)를 사용합니다. 예를 들어, 다음 일반 표현식에는 digits라는 식별자가 지정된 명명된 그룹이 포함되어 있습니다.

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

exec() 메서드를 사용하면 일치하는 명명된 그룹이 result 배열의 속성으로 추가됩니다.

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

다음 예제에서는 각각 name과 dom이라는 식별자가 지정된 두 개의 명명된 그룹을 사용합니다.

```
var emailPattern:RegExp =
  /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

예외 명명된 그룹은 ECMAScript 언어 사양에 포함되어 있지 않으며 ActionScript 3.0에 추가된 기능입니다.

플래그 및 속성

다음 표에서는 일반 표현식에 설정할 수 있는 다섯 개의 플래그에 대해 설명합니다. 각 플래그는 일반 표현식 객체의 속성으로 액세스할 수 있습니다.

플래그	속성	설명
g	global	둘 이상의 일치 항목을 찾습니다.
i	ignoreCase	일치하는 항목을 찾을 때 대/소문자를 구분하지 않습니다. 또한 A-Z 및 a-z 문자에는 적용되지만 É 및 é와 같은 확장 문자에는 적용되지 않습니다.
m	multiline	이 플래그가 설정되어 있는 경우 $와 ^은 각각 행의 시작 부분과 끝 부분을 찾을 수 있습니다.
s	dotall	이 플래그가 설정되어 있는 경우 .(도트)는 개행 문자(\n)와 일치시킬 수 있습니다.
x	extended	확장 일반 표현식을 사용할 수 있습니다. 패턴의 일부로 무시되는 공백을 일반 표현식에 입력할 수 있으며 이를 통해 일반 표현식 코드를 읽기 쉽게 입력할 수 있습니다.

이러한 속성은 읽기 전용이며 다음과 같이 일반 표현식 변수를 설정할 때 플래그(g, i, m, s, x)를 설정할 수 있습니다.

```
var re:RegExp = /abc/gimsx;
```

하지만 명명된 속성을 직접 설정할 수는 없습니다. 예를 들어, 다음 코드를 실행하면 오류가 발생합니다.

```
var re:RegExp = /abc/;
re.global = true; // 오류가 발생합니다 .
```

기본적으로 일반 표현식 선언에 플래그를 지정하지 않으면 해당 플래그가 설정되지 않으며 해당 속성이 false로 설정됩니다.

또한 다음과 같이 일반 표현식의 다른 두 속성도 사용할 수 있습니다.

- lastIndex 속성은 일반 표현식의 exec() 또는 test() 메서드를 다음에 호출할 때 사용하기 위해 문자열의 인덱스 위치를 지정합니다.
- source 속성은 일반 표현식의 패턴 부분을 정의하는 문자열을 지정합니다.

g(global) 플래그

g(global) 플래그가 포함되지 *않은* 일반 표현식은 일치 항목을 하나만 찾습니다. 예를 들어, g 플래그가 일반 표현식에 포함되어 있지 않은 경우 String.match() 메서드는 일치하는 하위 문자열을 하나만 반환합니다.

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // 출력 : she
```

g 플래그가 설정되면 Sting.match() 메서드는 다음과 같이 일치 항목을 여러 개 반환합니다.

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// 동일한 패턴이지만 이번에는 g 플래그 IS 가 설정되었습니다 .
trace(str.match(pattern)); // 출력 : she,shells,shore
```

i(ignoreCase) 플래그

기본적으로 일반 표현식으로 일치 항목을 찾을 때는 대/소문자를 구분합니다. 그러나 i(ignoreCase) 플래그를 설정하면 대/소문자 구분 특성이 무시됩니다. 예를 들어, 일반 표현식에 소문자 s를 포함하면 문자열의 첫 번째 문자인 대문자 S를 찾을 수 없습니다.

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // 출력 : 13 - 첫 문자 아님
```

그러나 i 플래그를 설정하면 일반 표현식에서 대문자 S를 찾습니다.

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // 출력 : 0
```

i 플래그는 A-Z 및 a-z 문자에 대해서만 대/소문자 구분 특성을 무시하고 É와 é 같은 확장 문자에는 적용되지 않습니다.

m(multiline) 플래그

m(multiline) 플래그가 설정되어 있지 않은 경우 ^은 문자열의 시작 부분을 찾고 $는 문자열의 끝 부분을 찾습니다. m 플래그가 설정되어 있는 경우 이러한 문자는 각각 행의 시작 부분과 끝 부분을 찾습니다. 개행 문자가 포함된 다음 문자열을 검토하십시오.

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // 문자열 시작 부분의 단어를 찾습니다 .
```

일반 표현식에 g(global) 플래그가 설정되어 있더라도 문자열의 시작 지점인 ^에 대해서는 일치 항목이 하나뿐이므로 match() 메서드는 하위 문자열을 하나만 찾습니다. 따라서 출력 결과는 다음과 같습니다.

```
Test
```

다음은 같은 코드에 m 플래그를 설정한 경우입니다.

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // 줄 시작 부분의 단어를 찾습니다 .
```

그러면 두 행의 시작 부분에 있는 단어가 모두 출력에 포함됩니다.

```
Test,Multiline
```

행의 끝을 나타내는 문자는 \n뿐이며 다음 문자로는 행의 끝을 나타낼 수 없습니다.

- 캐리지 리턴(\r) 문자
- 유니코드 행 분리 기호(\u2028) 문자
- 유니코드 단락 분리 기호(\u2029) 문자

s(dotall) 플래그

s(dotall 또는 "dot all") 플래그가 설정되어 있지 않은 경우 일반 표현식 패턴에 도트(.)를 포함해도 개행 문자(\n)를 찾을 수 없습니다. 다음 예제의 경우 일치하는 항목이 없습니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

하지만 s 플래그를 설정하면 개행 문자를 찾을 수 있습니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

이 경우 개행 문자를 포함하여 <p> 태그 내에 있는 전체 하위 문자열이 검색됩니다.

```
<p>Test
Multiline</p>
```

x(extended) 플래그

메타심볼이나 메타시퀀스가 많이 포함된 일반 표현식은 쉽게 읽을 수 없습니다. 예를 들면 다음과 같습니다.

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

일반 표현식에 x(extended) 플래그를 사용하면 일반 표현식 패턴에 입력하는 공백이 무시됩니다. 예를 들어, 다음 일반 표현식은 위의 예제에 나오는 일반 표현식과 동일합니다.

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

x 플래그를 설정한 경우 공백 문자를 찾지 않으려면 공백 앞에 백슬래시를 추가하십시오. 예를 들어, 다음 두 일반 표현식은 동일합니다.

```
/foo bar/
/foo \ bar/x
```

lastIndex 속성

lastIndex 속성은 문자열에서 다음 검색을 시작할 인덱스 위치를 지정합니다. 이 속성은 g 플래그가 true로 설정된 일반 표현식에서 호출되는 exec() 및 test() 메서드에 영향을 줍니다. 예를 들어 다음 코드를 검토하십시오.

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
  trace(pattern.lastIndex);
  result = pattern.exec(str);
}
```

lastIndex 속성은 기본적으로 0으로 설정되는데, 이렇게 하면 문자열의 맨 앞에서 검색을 시작합니다. 각 일치 항목을 찾은 후에는 이 속성이 해당 항목 다음의 인덱스 위치로 설정됩니다. 따라서 위의 코드를 실행하면 다음과 같이 출력됩니다.

```
0
5
11
18
25
36
44
```

global 플래그가 false로 설정되어 있는 경우 exec() 및 test() 메서드는 lastIndex 속성을 사용하거나 설정하지 않습니다.

String 클래스의 match(), replace() 및 search() 메서드는 해당 메서드를 호출할 때 사용된 일반 표현식의 lastIndex 속성 설정에 관계없이 문자열의 맨 앞에서 모든 검색을 시작합니다. 그러나 match() 메서드는 lastIndex 속성을 0으로 설정합니다.

lastIndex 속성을 설정하여 문자열에서 일반 표현식으로 일치 항목을 검색할 시작 위치를 조정할 수 있습니다.

source 속성

source 속성은 일반 표현식의 패턴 부분을 정의하는 문자열을 지정합니다. 예를 들면 다음과 같습니다.

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

문자열에 일반 표현식을 사용하는 데 필요한 메서드

RegExp 클래스에는 exec()와 test()라는 두 메서드가 포함되어 있습니다.

문자열에서 일반 표현식을 사용하여 일치 항목을 찾을 때는 RegExp 클래스의 exec() 및 test() 메서드뿐만 아니라 String 클래스에 포함된 match(), replace(), search() 및 splice() 메서드도 사용할 수 있습니다.

test() 메서드

RegExp 클래스의 test() 메서드는 다음 예제와 같이 제공된 문자열에 일반 표현식에 대해 일치하는 항목이 포함되어 있는지 여부만 확인합니다.

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // 출력 : true
```

exec() 메서드

RegExp 클래스의 exec() 메서드는 제공된 문자열에 일반 표현식에 대해 일치하는 항목이 포함되어 있는지 확인한 후 다음 항목이 포함된 배열을 반환합니다.

- 일치하는 하위 문자열
- 일반 표현식의 괄호 그룹에 대해 일치하는 하위 문자열

이 배열에는 하위 문자열 일치 항목의 시작 인덱스 위치를 나타내는 index 속성도 포함되어 있습니다.

예를 들어, 다음과 같은 코드를 살펴봅니다.

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7  -  415-555-1212
```

일반 표현식에 g(global) 플래그가 설정된 경우 하위 문자열을 여러 개 찾으려면 exec() 메서드를 여러 번 사용하십시오.

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
  trace(result.index, "\t", pattern.lastIndex, "\t", result);
  result = pattern.exec(str);
}
// 출력 :
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

RegExp 매개 변수를 사용하는 String 메서드

String 클래스의 match(), replace(), search() 및 split() 메서드는 일반 표현식을 매개 변수로 사용합니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 “문자열의 패턴 찾기 및 하위 문자열 바꾸기”를 참조하십시오.

예제: Wiki 파서

다음에 나오는 간단한 Wiki 텍스트 변환 예제에서는 다양한 일반 표현식 사용 방법을 보여 줍니다.

- 소스 Wiki 패턴과 일치하는 텍스트 행을 적절한 HTML 출력 문자열로 변환
- 일반 표현식을 사용하여 URL 패턴을 HTML `<a>` 하이퍼링크 태그로 변환
- 일반 표현식을 사용하여 미국 달러 문자열(예: `"$9.95"`)을 유로 문자열(예: `"8.24 €"`)로 변환

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. WikiEditor 응용 프로그램 파일은 Samples/WikiEditor 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
WikiEditor.mxml 또는 WikiEditor.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/regExpExamples/WikiParser.as	일반 표현식을 사용하여 Wiki 입력 텍스트 패턴을 동일한 HTML 출력으로 변환하는 메서드가 포함된 클래스입니다.
com/example/programmingas3/regExpExamples/URLParser.as	일반 표현식을 사용하여 URL 문자열을 HTML `<a>` 하이퍼링크 태그로 변환하는 메서드가 포함된 클래스입니다.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	일반 표현식을 사용하여 미국 달러 문자열을 유로 문자열로 변환하는 메서드가 포함된 클래스입니다.

WikiParser 클래스 정의

WikiParser 클래스에는 Wiki 입력 텍스트를 동일한 HTML 출력으로 변환하는 메서드가 포함되어 있습니다. 이 클래스는 강력한 기능의 Wiki 변환 응용 프로그램은 아니지만 일반 표현식을 사용하여 패턴 일치 및 문자열 변환을 수행하는 몇 가지 방법을 자세히 보여 줍니다.

생성자 함수는 다음과 같이 `setWikiData()` 메서드와 함께 Wiki 입력 텍스트의 샘플 문자열을 초기화합니다.

```
public function WikiParser()
{
  wikiData = setWikiData();
}
```

사용자가 샘플 응용 프로그램에서 [테스트] 버튼을 클릭하면 응용 프로그램에서 WikiParser 객체의 `parseWikiString()` 메서드를 호출하고, 이 메서드는 결과 HTML 문자열을 차례로 어셈블하는 다른 여러 메서드를 호출합니다.

```
public function parseWikiString(wikiString:String):String
{
  var result:String = parseBold(wikiString);
  result = parseItalic(result);
  result = linesToParagraphs(result);
  result = parseBullets(result);
  return result;
}
```

호출된 parseBold(), parseItalic(), linesToParagraphs() 및 parseBullets() 메서드는 각각 문자열의 replace() 메서드를 사용하여 일반 표현식으로 정의된 일치 패턴을 바꿉니다. 이를 통해 Wiki 입력 텍스트를 HTML 포맷 텍스트로 변환할 수 있습니다.

굵은체 및 기울임체 패턴 변환

parseBold() 메서드는 다음과 같이 Wiki 굵은체 텍스트 패턴(예: '''foo''')을 검색한 후 동일한 HTML 형식(예: foo)으로 변환합니다.

```
private function parseBold(input:String):String
{
  var pattern:RegExp = /'''(.*?)'''/g;
  return input.replace(pattern, "<b>$1</b>");
}
```

일반 표현식의 (.?*) 부분은 정의된 두 ''' 패턴 사이에 있는 모든 문자(*)를 찾습니다. 이때 ? 한정 기호는 최장 일치가 수행되지 않도록 하는 역할을 합니다. 즉, '''aaa''' bbb '''ccc'''와 같은 문자열에 대해 첫 번째로 일치하는 문자열은 ''' 패턴으로 시작하고 끝나는 전체 문자열이 아니라 '''aaa'''가 됩니다.

일반 표현식의 괄호는 캡처 그룹을 정의하고 replace() 메서드는 대체 문자열에서 $1 코드를 사용하여 이 그룹을 참조합니다. 일반 표현식에 g(global) 플래그를 설정하고 replace() 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

parseItalic() 메서드는 parseBold() 메서드와 비슷하지만 기울임체 텍스트에 대한 구분 기호로 세 개가 아니라 두 개의 아포스트로피('')를 확인한다는 차이가 있습니다.

```
private function parseItalic(input:String):String
{
  var pattern:RegExp = /''(.*?)''/g;
  return input.replace(pattern, "<i>$1</i>");
}
```

불릿 패턴 변환

다음 예제와 같이 `parseBullet()` 메서드는 Wiki 불릿 행 패턴(예: `* foo`)을 검색한 후 동일한 HTML 형식(예: `<li>foo</li>`)으로 변환합니다.

```
private function parseBullets(input:String):String
{
  var pattern:RegExp = /^\*(.*)/gm;
  return input.replace(pattern, "<li>$1</li>");
}
```

일반 표현식의 맨 앞에 `^` 심볼이 있으면 행의 시작 위치를 찾으며, `m`(`multiline`) 플래그가 설정되어 있으면 `^` 심볼이 문자열의 시작 위치 및 행의 시작 위치에 대응됩니다.

`\*` 패턴은 별표 문자를 찾습니다. 이때 `*` 한정 기호 대신 리터럴 별표를 나타내기 위해 백슬래시가 사용됩니다.

일반 표현식의 괄호는 캡처 그룹을 정의하고 `replace()` 메서드는 대체 문자열에서 `$1` 코드를 사용하여 이 그룹을 참조합니다. 일반 표현식에 `g`(`global`) 플래그를 설정하고 `replace()` 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

Wiki 단락 패턴 변환

`linesToParagraphs()` 메서드는 Wiki 입력 문자열의 각 행을 HTML `<p>` 단락 태그로 변환합니다. 메서드의 이러한 행은 Wiki 입력 문자열에서 빈 행을 제거합니다.

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

일반 표현식에 `^` 및 `$` 심볼이 있으면 행의 시작 및 끝 위치를 찾으며, `m`(`multiline`) 플래그가 설정되어 있으면 `^` 심볼이 문자열의 시작 위치 및 행의 시작 위치에 대응됩니다.

`replace()` 메서드는 일치하는 모든 하위 문자열(빈 행)을 빈 문자열(`""`)로 바꿉니다. 일반 표현식에 `g`(`global`) 플래그를 설정하고 `replace()` 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

URL을 HTML <a> 태그로 변환

사용자가 `urlToATag` 체크 상자를 선택한 경우 샘플 응용 프로그램에서 [Test] 버튼을 클릭하면 응용 프로그램에서 `URLParser.urlToATag()` 정적 메서드를 호출하여 Wiki 입력 문자열의 URL 문자열을 HTML `<a>` 태그로 변환합니다.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart,
                    "ig");
var result:String = input.replace(urlPattern,
                "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

RegExp() 생성자 함수는 다양한 구성 요소에서 일반 표현식(urlPattern)을 어셈블하는 데 사용됩니다. 이러한 구성 요소는 일반 표현식 패턴의 항목을 정의하는 각 문자열입니다.

protocol 문자열로 정의된 일반 표현식 패턴의 첫 번째 항목은 URL 프로토콜(http:// 또는 ftp://)을 정의합니다. 괄호는 ? 심볼로 나타내는 비캡처 그룹을 정의합니다. 즉, 괄호는 | 패턴의 그룹을 정의하는 데만 사용되며 이 그룹은 replace() 메서드의 대체 문자열에서 역참조 코드($1, $2, $3)와 일치하지 않습니다.

일반 표현식의 다른 구성 요소는 각각 일반 표현식 패턴에서 괄호로 표시되는 캡처 그룹을 사용하고 이러한 그룹은 replace() 메서드의 대체 문자열에서 역참조 코드($1, $2, $3)에 사용됩니다.

urlPart 문자열에 의해 정의된 패턴 부분은 a-z, 0-9, _ 또는 - 문자 중 *최소한* 하나 이상을 찾습니다. + 한정 기호는 최소한 하나 이상의 문자를 찾는다는 것을 나타내고 \.는 필수 도트(.) 문자를 나타냅니다. 또한 나머지 항목은 a-z, 0-9, _ 또는 - 문자 중 최소한 하나 이상의 다른 문자열을 찾습니다.

optionalUrlPart 문자열에 의해 정의된 패턴 부분은 도트(.) 문자와 모든 영숫자 문자(_ 및 - 포함)가 차례로 나오는 항목을 *0개 이상* 찾습니다. * 한정 기호는 0개 이상의 문자를 찾는다는 것을 나타냅니다.

replace() 메서드를 호출하면 일반 표현식이 사용되고 역참조를 통해 대체 HTML 문자열이 어셈블됩니다.

그러면 urlToATag() 메서드에서 emailToATag() 메서드를 호출합니다. 이 메서드는 비슷한 방법을 사용하여 전자 메일 패턴을 HTML <a> 하이퍼링크 문자열로 바꿉니다. 이 샘플 파일에서는 사용자의 이해를 돕기 위해 매우 간단한 일반 표현식을 제공하여 HTTP, FTP 및 전자 메일 URL을 찾습니다. 그러나 이러한 URL을 정확하게 찾기 위해 사용되는 일반 표현식은 훨씬 복잡합니다.

미국 달러 문자열을 유로 문자열로 변환

사용자가 dollarToEuro 체크 상자를 선택한 경우 샘플 응용 프로그램에서 [Test] 버튼을 클릭하면 응용 프로그램에서 CurrencyConverter.usdToEuro() 정적 메서드를 호출하여 미국 달러 문자열(예: "$9.95")을 유로 문자열(예: "8.24 €")로 변환합니다.

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

첫 번째 행은 미국 달러 문자열을 찾기 위한 간단한 패턴을 정의합니다. 이때 $ 문자 앞에 백슬래시(\) 이스케이프 문자가 추가됩니다.

replace() 메서드는 일반 표현식을 패턴 매처로 사용하고 usdStrToEuroStr() 함수를 호출하여 대체 문자열, 즉 유로 값을 확인합니다.

함수 이름이 replace() 메서드의 두 번째 매개 변수로 사용되는 경우에는 호출된 함수에 다음 항목이 매개 변수로 전달됩니다.

- 문자열의 일치 부분
- 캡처한 괄호 그룹 일치 항목. 이런 방식으로 전달되는 인수의 수는 캡처한 괄호 그룹 일치 항목의 수에 따라 다릅니다. 함수 코드 내에서 `arguments.length - 3`을 확인하면 캡처한 괄호 그룹 일치 항목의 수를 알 수 있습니다.
- 문자열에서 검색을 시작할 인덱스 위치
- 전체 문자열

`usdStrToEuroStr()` 메서드는 다음과 같이 미국 달러 문자열 패턴을 유로 문자열로 변환합니다.

```
private function usdToEuro(...args):String
{
  var usd:String = args[1];
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.828017;
  var euro:Number = Number(usd) * exchangeRate;
  trace(usd, Number(usd), euro);
  const euroSymbol:String = String.fromCharCode(8364); // €
  return euro.toFixed(2) + " " + euroSymbol;
}
```

`args[1]`은 `usdPrice` 일반 표현식을 사용하여 캡처한 괄호 그룹을 나타냅니다. 이는 미국 달러 문자열의 숫자 부분, 즉 `$` 심볼이 없는 달러 금액 부분입니다. 이 메서드는 환율 변환을 적용하고 결과 문자열(앞의 $ 심볼 대신 뒤에 `€` 심볼 추가)을 반환합니다.

제 10 장

10 이벤트 처리

이벤트 처리 시스템을 사용하면 프로그래머가 편리한 방식으로 사용자 입력 및 시스템 이벤트에 응답할 수 있습니다. ActionScript 3.0 이벤트 모델은 사용이 편리할 뿐만 아니라 표준 규격을 준수하며 Adobe Flash Player 9 표시 목록과 잘 통합됩니다. 새 이벤트 모델은 산업 표준 이벤트 처리 아키텍처인 DOM(Document Object Model) 레벨 3 이벤트 사양을 기초로 ActionScript 프로그래머를 위해 강력하고 직관적인 이벤트 처리 도구를 제공합니다.

이 장은 다섯 개의 단원으로 구성되어 있는데, 처음 두 단원에서는 ActionScript의 이벤트 처리에 대한 배경 정보를 설명하고 나머지 세 단원에서는 이벤트 모델의 기본 개념인 이벤트 흐름, 이벤트 객체, 이벤트 리스너에 대해 각각 설명합니다. ActionScript 3.0 이벤트 처리 시스템은 표시 목록과 밀접하게 상호 작용하므로 이 장에서는 사용자가 표시 목록에 대해 기본적인 내용을 이해하고 있다고 가정합니다. 자세한 내용은 349페이지의 “디스플레이 프로그래밍”을 참조하십시오.

목차

이벤트 처리의 기초

이벤트 처리 소개

이벤트는 SWF 파일에서 발생하는 동작이며 프로그래머에게 매우 중요합니다. 예를 들어, 대부분의 SWF 파일은 사용자 상호 작용을 지원합니다. 이러한 상호 작용은 마우스 클릭에 응답하는 것처럼 간단할 수도 있고 양식에 입력된 데이터를 적용 및 처리하는 것처럼 복잡할 수도 있습니다. SWF 파일과의 이러한 상호 작용을 이벤트라고 합니다. 그러나 서버에서 데이터 로드를 마쳤을 때 또는 연결된 카메라가 활성화될 때와 같이 직접적인 사용자 상호 작용 없이도 이벤트가 발생할 수 있습니다.

ActionScript 3.0에서 각 이벤트는 Event 클래스 또는 해당 하위 클래스의 인스턴스인 이벤트 객체로 나타냅니다. 이벤트 객체는 특정 이벤트에 대한 정보를 저장할 뿐만 아니라 해당 이벤트 객체를 쉽게 조작하는 데 사용할 수 있는 메서드도 포함합니다. 예를 들어, Flash Player에서는 마우스 클릭을 감지하면 특정 마우스 클릭 이벤트를 나타내는 이벤트 객체(MouseEvent 클래스의 인스턴스)가 만들어집니다.

이벤트 객체를 만든 후에는 Flash Player에서 이 이벤트 객체를 이벤트 대상 객체에 *전달합니다*. 전달되는 이벤트 객체의 대상으로 사용되는 객체를 *이벤트 대상*이라고 합니다. 예를 들어, 연결된 카메라가 활성화되면 Flash Player에서 이벤트 객체를 직접 이벤트 대상에 전달하는데, 이 경우 이벤트 대상은 해당 카메라를 나타내는 객체입니다. 그러나 이벤트 대상이 표시 목록에 있는 경우에는 표시 목록 계층 구조를 통해 해당 표시 목록에서 이벤트 대상을 찾을 때까지 이벤트 객체가 전달됩니다. 경우에 따라서는 이벤트 객체가 같은 경로를 따라 표시 목록 계층 구조에서 다시 위로"비블링"될 수도 있습니다. 이러한 표시 목록 계층 구조 순회를 *이벤트 흐름*이라고 합니다.

코드에서 이벤트 리스너를 사용하여 이벤트 객체를 "수신"할 수 있습니다. *이벤트 리스너*는 특정 이벤트에 응답하기 위해 작성하는 함수 또는 메서드입니다. 프로그램에서 이벤트에 응답할 수 있도록 하려면 이벤트 대상 또는 이벤트 객체의 이벤트 흐름에 포함된 표시 목록 객체에 이벤트 리스너를 추가해야 합니다.

이벤트 리스너 코드를 작성할 때는 다음과 같은 기본 구조를 따릅니다. 여기서 굵게 표시된 요소는 특정한 경우에 입력하는 자리 표시자입니다.

```
function eventResponse(eventObject:EventType):void
{
  // 이벤트에 응답하여 수행할 액션을 여기에 입력하십시오 .
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

이 코드는 다음과 같은 두 가지 작업을 수행합니다. 먼저 함수를 정의하여 이벤트에 대한 응답으로 수행할 작업을 지정할 수 있습니다. 그런 다음 소스 객체의 addEventListener() 메서드를 호출하여 지정된 이벤트에 대해 함수를 "제공함으로써" 이벤트 발생 시 함수 작업이 수행되도록 합니다. 실제로 이벤트가 발생하면 이벤트 대상이 이벤트 리스너로 등록된 모든 함수와 메서드 목록을 확인합니다. 그런 다음 각 함수와 이벤트를 차례로 호출하여 이벤트 객체를 매개 변수로 전달합니다.

이벤트 리스너를 직접 만들려면 이 코드에서 네 가지를 변경해야 합니다. 먼저 함수 이름을 사용자가 사용할 이름으로 변경해야 합니다. 코드에서 eventResponse라고 표시된 두 군데를 변경합니다. 두 번째로, 수신할 이벤트에서 전달한 이벤트 객체에 적절한 클래스 이름(코드에서는 EventType)을 지정해야 하며 특정 이벤트에 적절한 상수(샘플의 EVENT_NAME)를 지정해야 합니다. 세 번째로, 이벤트를 전달할 객체에서 addEventListener() 메서드(이 코드에서는 eventTarget)를 호출해야 합니다. 필요에 따라 함수의 매개 변수로 사용된 변수의 이름(이 코드에서는 eventObject)을 변경할 수 있습니다.

일반적인 이벤트 처리 작업

이 장에서 다루고 있는 일반적인 이벤트 처리 작업은 다음과 같습니다.

- 이벤트에 응답할 코드 작성
- 코드가 이벤트에 응답하지 않도록 중지
- 이벤트 객체 다루기
- 이벤트 흐름 다루기
  - 이벤트 흐름 정보 확인
  - 이벤트 흐름 중단
  - 기본 비헤이비어 방지
- 클래스에서 이벤트 전달
- 사용자 정의 이벤트 유형 만들기

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 기본 비헤이비어: 일부 이벤트에는 일반적으로 이벤트와 함께 발생하는 비헤이비어가 포함되어 있는데, 이를 기본 비헤이비어라고 합니다. 예를 들어, 사용자가 텍스트 필드에 텍스트를 입력하면 텍스트 입력 이벤트가 발생합니다. 이 이벤트의 기본 비헤이비어는 텍스트 필드에 입력한 문자를 실제로 표시하는 것이지만, 어떠한 이유로든 입력한 문자를 표시하지 않으려는 경우 이 기본 비헤이비어를 재정의할 수 있습니다.
- 전달: 이벤트가 발생했음을 이벤트 리스너에 알리는 것입니다.
- 이벤트: 객체에 발생하는 것으로, 객체가 다른 객체에 알릴 수 있습니다.

- 이벤트 흐름: 표시 목록의 객체(화면에 표시되는 객체)에 이벤트가 발생하면 해당 객체를 포함하는 모든 객체에 이벤트를 알린 후 이벤트 리스너에 알립니다. 이 프로세스는 Stage에서 시작하여 표시 목록을 통해 이벤트가 발생한 실제 객체로 이동한 다음 다시 Stage로 돌아갑니다. 이 프로세스를 이벤트 흐름이라고 합니다.
- 이벤트 객체: 특정 이벤트의 발생에 대한 정보가 포함된 객체입니다. 이 객체는 이벤트를 전달할 때 모든 리스너에 전송됩니다.
- 이벤트 대상: 이벤트를 실제로 전달하는 객체입니다. 예를 들어, 사용자가 Stage 안에 있는 Sprite 안의 버튼을 클릭하면 해당하는 모든 객체가 이벤트를 전달하지만 이벤트 대상은 실제로 이벤트가 발생한 객체(이 경우, 클릭한 버튼)입니다.
- 리스너: 특정 이벤트가 발생하는 경우 통지되어야 함을 나타내기 위해 객체에 등록된 객체 또는 함수입니다.

이 장의 예제를 사용한 작업

장의 내용을 단계적으로 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 기본적으로 이 장의 모든 코드 샘플에는 코드 결과 테스트를 위해 `trace()` 함수 호출이 포함되어 있습니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널에서 코드 샘플의 `trace()` 함수에 대한 결과를 확인합니다.

일부 코드 샘플은 보다 복잡하며 클래스로 작성되어 있습니다. 이러한 예제를 테스트하려면:

1. 빈 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 새 ActionScript 파일을 만들고 Flash 문서와 같은 디렉토리에 저장합니다. 파일 이름은 코드 샘플에 있는 클래스 이름과 일치해야 합니다. 예를 들어, 코드 샘플에 EventTest라는 클래스가 정의된 경우 ActionScript 파일을 EventTest.as라는 이름으로 저장합니다.
3. ActionScript 파일에 코드 샘플을 복사하고 파일을 저장합니다.
4. Flash 문서에서 스테이지 또는 작업 영역의 빈 부분을 클릭하여 문서 속성 관리자를 활성화합니다.
5. 텍스트에서 복사한 ActionScript 클래스의 이름을 속성 관리자의 [문서 클래스] 필드에 입력합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널에서 예제 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

ActionScript 3.0과 이전 버전의 이벤트 처리 방식 비교

ActionScript 3.0과 이전 버전의 ActionScript에서 이벤트를 처리하는 방식 중 가장 큰 차이점은 ActionScript 3.0에는 이벤트 처리 시스템이 하나뿐인 반면 이전 버전의 ActionScript에는 이벤트 처리 시스템이 여러 개 있다는 것입니다. 이 단원에서는 먼저 이전 버전의 ActionScript에서 이벤트를 처리하는 방식에 대해 간략하게 설명한 다음, ActionScript 3.0에서 이벤트 처리 방식이 어떻게 변경되었는지 설명합니다.

이전 버전의 ActionScript에서 이벤트 처리

ActionScript 3.0 이전 버전의 ActionScript에서는 다음과 같은 다양한 방법을 사용하여 이벤트를 처리합니다.

- `on()` 이벤트 핸들러 - Button 및 MovieClip 인스턴스에 직접 배치할 수 있습니다.
- `onClipEvent()` 핸들러 - MovieClip 인스턴스에 직접 배치할 수 있습니다.
- 콜백 함수 속성 - `XML.onload` 및 `Camera.onActivity` 등이 있습니다.
- 이벤트 리스너 - `addListener()` 메서드를 사용하여 등록합니다.
- UIEventDispatcher 클래스 - DOM 이벤트 모델을 부분적으로 구현했습니다.

이러한 메커니즘에는 각각 고유한 장점과 단점이 있습니다. `on()` 및 `onClipEvent()` 핸들러는 쉽게 사용할 수 있지만 버튼과 무비 클립에 직접 배치한 코드를 찾기가 어려우므로 이후 프로젝트를 관리하는 데 어려움이 있습니다. 콜백 함수도 간단하게 구현할 수는 있지만 지정된 이벤트에 대해 콜백 함수를 하나만 사용할 수 있습니다. 이와 달리 이벤트 리스너는 리스너 객체와 함수를 만들어야 할 뿐만 아니라 이벤트를 생성하는 객체에 리스너를 등록해야 하므로 쉽게 구현할 수 없습니다. 그러나 이를 통해 리스너 객체를 여러 개 만들어 같은 이벤트에 모든 리스너를 등록할 수 있습니다.

ActionScript 2.0의 구성 요소 개발을 통해 다른 이벤트 모델이 생성되었습니다. UIEventDispatcher 클래스에 구현된 이 새 모델은 DOM 이벤트 사양의 하위 집합을 기초로 하므로 구성 요소 이벤트 처리 작업에 익숙한 개발자는 비교적 큰 어려움 없이 새 ActionScript 3.0 이벤트 모델을 사용할 수 있습니다.

다양한 이벤트 모델에 사용되는 구문은 여러 가지 면에서 공통되는 부분이 있지만 나머지 부분에서는 서로 다릅니다. 예를 들어, ActionScript 2.0에서는 `TextField.onChanged`와 같은 일부 속성을 콜백 함수 또는 이벤트 리스너로 사용할 수 있습니다. 그러나 리스너 객체 등록 구문은 리스너를 지원하는 여섯 개의 클래스 중 하나를 사용하는지 아니면 UIEventDispatcher 클래스를 사용하는지에 따라 다릅니다. Key, Mouse, MovieClipLoader, Selection, Stage 및 TextField 클래스의 경우 `addListener()` 메서드를 사용하지만 구성 요소 이벤트 처리의 경우에는 `addEventListener()` 메서드를 사용합니다.

또한 이벤트 처리 모델이 여러 개인 경우에는 사용된 메커니즘에 따라 이벤트 핸들러 함수의 범위가 크게 달라진다는 문제도 있습니다. 즉, `this` 키워드의 의미가 이벤트 처리 시스템 간에 일치하지 않습니다.

ActionScript 3.0의 이벤트 처리

ActionScript 3.0에서는 이전 버전의 언어에 사용된 다양한 이벤트 처리 메커니즘을 대체하는 단일 이벤트 처리 모델을 사용합니다. 새 이벤트 모델은 DOM(Document Object Model) 레벨 3 이벤트 사양을 기초로 합니다. SWF 파일 포맷은 Document Object Model 표준을 따르지는 않지만 표시 목록과 DOM 구조에는 DOM 이벤트 모델을 구현할 수 있도록 비슷한 점이 많습니다. 표시 목록의 객체는 DOM 계층 구조의 노드에 해당하므로 여기에서 *표시 목록 객체*와 *노드*라는 용어를 바꾸어 사용할 수 있습니다.

Flash Player에서 DOM 이벤트 모델을 구현하는 경우 기본 비헤이비어라는 개념이 포함됩니다. *기본 비헤이비어*는 특정 이벤트의 결과로 Flash Player에서 실행되는 액션입니다.

기본 비헤이비어

일반적으로 개발자는 이벤트에 응답하는 코드를 작성합니다. 그러나 경우에 따라서는 해당 이벤트와 일반적으로 연결된 비헤이비어가 있어 개발자가 비헤이비어 취소 코드를 추가하지 않는 한 Flash Player에서 자동으로 비헤이비어를 실행합니다. Flash Player에서 자동으로 실행하는 이러한 비헤이비어를 기본 비헤이비어라고 합니다.

예를 들어, 사용자가 TextField 객체에 텍스트를 입력하는 경우 일반적으로 이 텍스트가 TextField 객체에 표시될 것으로 예상하므로 이러한 비헤이비어가 Flash Player에 내장됩니다. 이 기본 비헤이비어가 발생하지 않도록 하려면 새 이벤트 처리 시스템을 사용하여 취소하면 됩니다. 보다 구체적으로 설명하면, 사용자가 TextField 객체에 텍스트를 입력하는 경우 Flash Player에서 TextEvent 클래스의 인스턴스를 만들어 사용자 입력을 표시합니다. Flash Player에서 TextField 객체에 텍스트를 표시하지 않도록 하려면 특정 TextEvent 인스턴스에 액세스하여 해당 인스턴스의 `preventDefault()` 메서드를 호출해야 합니다.

기본 비헤이비어 중 일부는 취소할 수 없습니다. 예를 들어, 사용자가 TextField 객체에서 단어를 두 번 클릭하면 Flash Player에서 MouseEvent 객체를 생성합니다. 이때 커서 아래의 단어가 강조 표시되는 기본 비헤이비어는 취소할 수 없습니다.

대부분의 이벤트 객체 유형에는 기본 비헤이비어가 연결되어 있지 않습니다. 예를 들어, 네트워크 연결이 설정되면 Flash Player에서 connect 이벤트 객체를 전달하지만 이 객체에는 기본 비헤이비어가 연결되어 있지 않습니다. Event 클래스 및 하위 클래스에 대한 API 설명서에서는 각 이벤트 유형 목록을 제공하고 해당 이벤트에 연결된 기본 비헤이비어에 대한 내용 및 각 비헤이비어를 취소할 수 있는지 여부를 설명합니다.

기본 비헤이비어는 Flash Player에서 전달하는 이벤트 객체에만 연결되고 ActionScript를 통해 프로그래밍 방식으로 전달되는 이벤트 객체에는 연결되지 않는다는 사실을 이해해야 합니다. 예를 들어, EventDispatcher 클래스의 메서드를 사용하면 `textInput` 유형의 이벤트 객체를 전달할 수 있지만 이 이벤트 객체에는 기본 비헤이비어가 연결되지 않습니다. 즉, Flash Player에서는 개발자가 프로그래밍 방식으로 전달한 `textInput` 이벤트의 결과로 TextField 객체에 문자를 표시하지 않습니다.

ActionScript 3.0 이벤트 리스너의 새로운 기능

ActionScript 2.0의 `addListener()` 메서드를 사용해 본 개발자는 ActionScript 2.0 이벤트 리스너 모델과 ActionScript 3.0 이벤트 모델의 차이점을 쉽게 파악할 수 있습니다. 다음 목록에서는 두 이벤트 모델의 몇 가지 주요 차이점을 설명합니다.

- ActionScript 2.0에서는 이벤트 리스너를 추가할 때 상황에 따라 `addListener()` 또는 `addEventListener()`를 사용하지만 ActionScript 3.0에서는 항상 `addEventListener()`를 사용합니다.
- ActionScript 2.0에는 이벤트 흐름이 없으므로 이벤트를 브로드캐스팅하는 객체에서만 `addListener()` 메서드를 호출할 수 있지만 ActionScript 3.0에서는 이벤트 흐름에 포함된 모든 객체에서 `addEventListener()` 메서드를 호출할 수 있습니다.
- ActionScript 2.0에서는 함수, 메서드 또는 객체를 이벤트 리스너로 사용할 수 있지만 ActionScript 3.0에서는 함수 또는 메서드만 이벤트 리스너로 사용할 수 있습니다.

이벤트 흐름

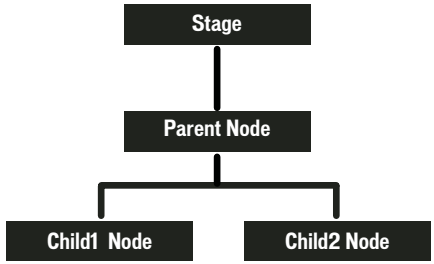
Flash Player에서는 이벤트가 발생할 때마다 이벤트 객체를 전달합니다. 이때 이벤트 대상이 표시 목록에 없으면 이벤트 객체를 이벤트 대상에 직접 전달합니다. 예를 들어, Flash Player에서는 progress 이벤트 객체를 URLStream 객체에 직접 전달합니다. 그러나 이벤트 대상이 표시 목록에 있는 경우에는 이벤트 객체를 표시 목록에 전달하고 이 이벤트 객체는 해당 이벤트 대상을 찾을 때까지 표시 목록을 순회합니다.

*이벤트 흐름*은 이벤트 객체가 표시 목록을 이동하는 방식을 기술합니다. 표시 목록은 트리로 나타낼 수 있는 계층 구조로 구성되어 있으며, 표시 목록 계층 구조의 맨 위에는 Stage, 즉 표시 목록의 루트로 사용되는 특수한 표시 객체 컨테이너가 있습니다. Stage는 flash.display.Stage 클래스로 나타내고 표시 객체를 통해서만 액세스할 수 있습니다. 모든 표시 객체에는 해당 응용 프로그램의 Stage를 참조하는 `stage` 속성이 있습니다.

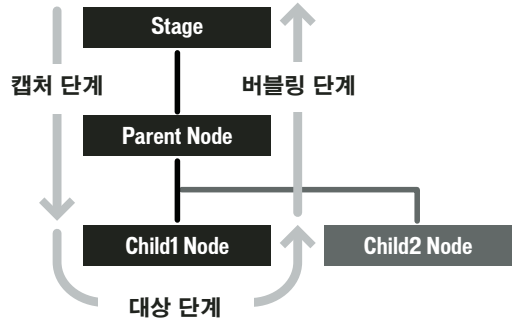
Flash Player에서 이벤트 객체를 전달하는 경우 해당 이벤트 객체는 Stage에서 *대상 노드*까지 포함하는 라운드 트립을 만듭니다. DOM 이벤트 사양에서는 대상 노드를 이벤트 대상을 나타내는 노드로 정의합니다. 즉, 대상 노드는 이벤트가 발생한 표시 목록 객체입니다. 예를 들어, 사용자가 `child1`이라는 표시 목록 객체를 클릭하면 Flash Player에서 `child1`을 대상 노드로 사용하여 이벤트 객체를 전달합니다.

이벤트 흐름은 개념적으로 세 부분으로 구분됩니다. 첫 번째 부분은 캡처 단계라고 하고 Stage에서 대상 노드의 부모까지 모든 노드를 포함합니다. 두 번째 부분은 대상 단계라고 하고 대상 노드만 포함합니다. 마지막으로 세 번째 부분은 버블링 단계라고 하고 대상 노드의 부모에서 다시 Stage로 돌아오는 과정에서 만나게 되는 노드를 포함합니다.

다음 다이어그램과 같이 표시 목록을 Stage가 맨 위에 있는 세로 계층 구조로 생각해 보면 각 단계의 이름을 쉽게 이해할 수 있습니다.



사용자가 `Child1 Node`를 클릭하면 Flash Player에서는 이벤트 객체를 이벤트 흐름에 전달합니다. 다음 그림과 같이 객체는 `Stage`에서 시작하여 아래에 있는 `Parent Node`와 `Child1 Node`로 차례로 이동한 다음, 다시 `Stage`까지 위로 "버블링"됩니다. 버블링 단계에서 `Stage`로 이동할 때는 다시 `Parent Node`를 통과합니다.



이 예제에서 캡처 단계는 처음에 아래로 이동하는 동안 `Stage`와 `Parent Node`를 포함하고 대상 단계는 `Child1 Node`에서 소비한 시간을 포함합니다. 마지막으로 버블링 단계는 다시 루트 노드로 이동하는 동안 만나는 `Parent Node`와 `Stage`를 포함합니다.

이벤트 흐름 덕분에 ActionScript 프로그래머는 이전에 비해 더욱 강력한 이벤트 처리 시스템을 사용할 수 있습니다. 이전 버전의 ActionScript에는 이벤트 흐름이 없으므로 이벤트를 생성하는 객체에만 이벤트 리스너를 추가할 수 있습니다. 그러나 ActionScript 3.0에서는 대상 노드뿐만 아니라 이벤트 흐름을 따라 모든 노드에 이벤트 리스너를 추가할 수 있습니다.

이벤트 흐름을 따라 이벤트 리스너를 추가할 수 있는 기능은 사용자 인터페이스 구성 요소에 객체가 두 개 이상 포함되어 있는 경우에 유용합니다. 예를 들어, 버튼 객체에 버튼 레이블로 사용되는 텍스트 객체가 포함될 수 있습니다. 이벤트 흐름에 리스너를 추가하는 기능이 없는 경우 버튼에서 발생하는 모든 클릭 이벤트에 대한 알림을 받으려면 버튼 객체와 텍스트 객체 둘 다에 리스너를 추가해야 합니다. 그러나 이벤트 흐름 기능을 사용하면 텍스트 객체 또는 텍스트 객체에 의해 식별되는 버튼 객체 영역에서 발생하는 클릭 이벤트를 처리하는 단일 이벤트 리스너를 버튼 객체에 추가할 수 있습니다.

그러나 이벤트 흐름의 세 단계 중 일부 단계에 참여하지 못하는 이벤트 객체도 있습니다. `enterFrame` 및 `init` 이벤트 유형 등 일부 유형의 이벤트는 대상 노드에 직접 전달되고 캡처 단계 및 버블링 단계에 참여하지 않습니다. Socket 클래스의 인스턴스에 전달되는 이벤트와 같이 표시 목록에 없는 객체를 대상으로 하는 이벤트도 있습니다. 이러한 이벤트 객체도 캡처 및 버블링 단계에 참여하지 않고 대상 객체에 직접 전달됩니다.

특정 이벤트 유형의 동작 방식을 확인하려면 API 설명서를 확인하거나 이벤트 객체의 속성을 검토하십시오. 이벤트 객체의 속성을 검토하는 데 대한 자세한 내용은 다음 단원에서 설명합니다.

이벤트 객체

이벤트 객체는 새 이벤트 처리 시스템에서 크게 두 가지 목적으로 사용됩니다. 첫 번째로 이벤트 객체는 속성 집합에 특정 이벤트에 대한 정보를 저장하여 실제 이벤트를 나타내고, 두 번째로 이벤트 객체에는 이벤트 객체를 조작하고 이벤트 처리 시스템의 비헤이비어에 영향을 줄 수 있는 메서드 집합이 포함되어 있습니다.

이러한 속성과 메서드에 쉽게 액세스할 수 있도록 Flash Player API에서는 모든 이벤트 객체의 기본 클래스로 사용되는 Event 클래스를 정의합니다. Event 클래스는 모든 이벤트 객체에 공통으로 사용되는 기본 속성 및 메서드 집합을 정의합니다.

이 단원에서는 먼저 Event 클래스 속성과 Event 클래스 메서드에 대해 차례로 설명하고 Event 클래스의 하위 클래스를 사용하는 이유에 대해 설명합니다.

Event 클래스 속성 이해

Event 클래스는 이벤트 객체에 대해 다음과 같은 중요한 정보를 제공하는 많은 읽기 전용 속성 및 상수를 정의합니다.

- 이벤트 객체 유형은 상수로 나타내고 `Event.type` 속성에 저장됩니다.
- 이벤트의 기본 비헤이비어를 취소할 수 있는지 여부는 부울 값으로 나타내고 `Event.cancelable` 속성에 저장됩니다.
- 이벤트 흐름 정보는 나머지 속성에 포함됩니다.

이벤트 객체 유형

모든 이벤트 객체에는 이벤트 유형이 연결되어 있으며 이벤트 유형은 Event.type 속성에 문자열 값으로 저장됩니다. 코드에서 유형이 다른 객체를 구별할 수 있도록 이벤트 객체의 유형을 알고 있으면 유용합니다. 예를 들어, 다음 코드에서는 clickHandler() 리스너 함수가 myDisplayObject에 전달되는 마우스 클릭 이벤트 객체에 응답하도록 지정합니다.

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Event 클래스 자체에 연결된 이벤트 유형은 24개 정도이며 각각 Event 클래스 상수로 나타냅니다. Event 클래스 정의에서 발췌한 다음 코드에서는 이러한 이벤트 유형 중 일부를 보여 줍니다.

```
package flash.events
{
  public class Event
  {
    // 클래스 상수
    public static const ACTIVATE:String = "activate";
    public static const ADDED:String    = "added";
    // 간단히 하기 위해 나머지 상수는 생략됨
  }
}
```

이러한 상수를 사용하면 특정 이벤트 유형을 쉽게 참조할 수 있습니다. 실제로 이벤트 유형을 나타내는 문자열 대신 상수를 사용해야 합니다. 코드에 상수 이름 철자를 잘못 입력하면 컴파일러에서 이러한 오류를 포착하지만 문자열을 사용하는 경우에는 컴파일할 때 철자 오류가 나타나지 않으므로 쉽게 디버깅할 수 없는 비헤이비어가 발생할 수 있습니다. 예를 들어, 이벤트 리스너를 추가할 때 다음 코드를 사용하십시오.

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

이때 다음 코드를 사용하지 않는 것이 좋습니다.

```
myDisplayObject.addEventListener("click", clickHandler);
```

기본 비헤이비어 정보

cancelable 속성에 액세스하여 특정 이벤트 객체의 기본 비헤이비어를 취소할 수 있는지 여부를 검사하도록 코드를 작성할 수 있습니다. cancelable 속성에는 기본 비헤이비어를 취소할 수 있는지 여부를 나타내는 부울 값이 포함되어 있습니다. preventDefault() 메서드를 사용하면 소수의 이벤트와 연결된 기본 비헤이비어를 취소할 수 있습니다. 자세한 내용은 307페이지의 “기본 이벤트 비헤이비어 취소”를 참조하십시오.

이벤트 흐름 정보

나머지 Event 클래스 속성에는 이벤트 객체 및 해당 이벤트 객체와 이벤트 흐름 간의 관계에 대한 중요 정보가 포함되어 있습니다(다음 목록의 설명 참조).

- `bubbles` 속성에는 해당 이벤트 객체가 참여하는 이벤트 흐름의 단계에 대한 정보가 포함됩니다.
- `eventPhase` 속성은 이벤트 흐름에서 현재 단계를 나타냅니다.
- `target` 속성에는 이벤트 대상에 대한 참조가 저장됩니다.
- `currentTarget` 속성에는 현재 이벤트 객체를 처리하고 있는 표시 목록 객체에 대한 참조가 저장됩니다.

bubbles 속성

이벤트 객체가 이벤트 흐름의 버블링 단계에 참여하는 경우, 즉 이벤트 객체가 대상 노드에서 다시 전달되어 조상 노드를 거쳐 Stage에 도달하는 경우 해당 이벤트가 버블링된다고 합니다. `Event.bubbles` 속성에는 이벤트 객체가 버블링 단계에 참여하는지 여부를 나타내는 부울 값이 저장됩니다. 버블링되는 모든 이벤트는 캡처 및 대상 단계에도 참여하므로 버블링되는 이벤트는 이벤트 흐름의 세 단계에 모두 참여하게 됩니다. 이 값이 `true`이면 이벤트 객체가 세 단계에 모두 참여하고, 이 값이 `false`이면 이벤트 객체가 버블링 단계에 참여하지 않습니다.

eventPhase 속성

`eventPhase` 속성을 검토하면 이벤트 객체의 이벤트 단계를 확인할 수 있습니다.
`eventPhase` 속성에는 이벤트 흐름의 세 단계 중 하나를 나타내는 부호 없는 정수 값이 포함됩니다. Flash Player API에서는 다음 코드와 같이 부호 없는 세 정수 값에 해당하는 세 개의 상수가 포함된 별도의 EventPhase 클래스를 정의합니다.

```
package flash.events
{
  public final class EventPhase
  {
    public static const CAPTURING_PHASE:uint = 1;
    public static const AT_TARGET:uint       = 2;
    public static const BUBBLING_PHASE:uint  = 3;
  }
}
```

이러한 상수는 `eventPhase` 속성의 유효한 세 값에 해당합니다. 상수를 사용하면 쉽게 읽을 수 있는 코드를 작성할 수 있습니다. 예를 들어, 이벤트 대상이 대상 단계에 있는 경우에만 `myFunc()` 함수를 호출하려면 다음 코드를 사용하여 이 조건을 테스트할 수 있습니다.

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
```

```
  myFunc();
}
```

target 속성

target 속성에는 이벤트 대상 객체에 대한 참조가 있습니다. 이는 경우에 따라 간단할 수 있는데, 예를 들어, 마이크가 활성화되는 경우 이벤트 객체의 대상은 Microphone 객체입니다. 그러나 대상이 표시 목록에 있는 경우에는 표시 목록 계층 구조를 고려해야 합니다. 예를 들어, 표시 목록 객체가 겹쳐 있는 지점에서 사용자가 마우스 클릭을 입력하면 Flash Player에서는 항상 Stage와 가장 멀리 떨어진 객체를 이벤트 대상으로 선택합니다.

복잡한 SWF 파일, 특히 크기가 작은 여러 개의 자식 객체로 장식된 버튼이 있는 SWF 파일의 경우 target 속성은 버튼이 아니라 버튼의 자식 객체를 가리키는 경우가 많으므로 자주 사용되지 않습니다. 이러한 경우 target 속성은 버튼의 자식 객체를 가리키지만 이 속성은 버튼을 가리키므로 일반적으로 버튼에 이벤트 리스너를 추가하고 currentTarget 속성을 사용하는 것이 좋습니다.

currentTarget 속성

currentTarget 속성에는 현재 이벤트 객체를 처리하고 있는 객체에 대한 참조가 포함됩니다. 검토 중인 이벤트 객체를 어떤 노드에서 처리하고 있는지 모른다는 것이 잘 이해가 되지 않을 수도 있지만 이벤트 객체의 이벤트 흐름에 있는 모든 표시 객체에 리스너 함수를 추가할 수 있으며 아무 위치에나 리스너 함수를 배치할 수 있음을 기억하십시오. 또한 같은 리스너 함수를 여러 표시 객체에 추가할 수도 있습니다. 프로젝트가 더 복잡해지고 크기가 커질수록 currentTarget 속성의 유용성도 커집니다.

Event 클래스 메서드 이해

Event 클래스 메서드는 다음과 같은 세 가지 범주로 구성되어 있습니다.

- 유틸리티 메서드 - 이벤트 객체의 복사본을 만들거나 문자열로 변환할 수 있습니다.
- 이벤트 흐름 메서드 - 이벤트 흐름에서 이벤트 객체를 제거합니다.
- 기본 비헤이비어 메서드 - 기본 비헤이비어를 취소하거나 취소 여부를 확인합니다.

Event 클래스 유틸리티 메서드

Event 클래스에는 두 가지 유틸리티 메서드가 있습니다. clone() 메서드를 사용하면 이벤트 객체의 복사본을 만들 수 있으며, toString() 메서드를 사용하면 이벤트 객체 속성의 문자열 표현과 해당 속성 값을 생성할 수 있습니다. 이러한 두 메서드는 모두 이벤트 모델 시스템에서 내부적으로 사용되지만 일반적인 용도로 개발자에게 노출됩니다.

Event 클래스의 하위 클래스를 만드는 고급 개발자의 경우 이벤트 하위 클래스가 제대로 작동하도록 두 유틸리티 메서드 버전을 모두 재정의하여 새로 구현해야 합니다.

이벤트 흐름 중단

Event.stopPropogation() 메서드 또는 Event.stopImmediatePropogation() 메서드를 호출하면 이벤트 흐름에서 이벤트 객체가 계속 진행되지 않도록 중단할 수 있습니다. 두 메서드는 다음과 같이 현재 노드의 다른 이벤트 리스너를 실행할 수 있는지 여부만 다르고 나머지는 거의 같습니다.

- Event.stopPropogation() 메서드를 사용하면 현재 노드에서 다른 이벤트 리스너가 실행된 경우에만 이벤트 객체가 다음 노드로 이동되지 않습니다.
- Event.stopImmediatePropogation() 메서드를 사용하면 이벤트 객체가 다음 노드로 이동되지 않지만 현재 노드에서 다른 이벤트 리스너를 실행할 수 없습니다.

두 메서드 중 하나를 호출해도 이벤트와 연결된 기본 비헤이비어의 발생 여부에는 아무 영향을 주지 않습니다. 기본 비헤이비어를 취소하려면 Event 클래스의 기본 비헤이비어 메서드를 사용하십시오.

기본 이벤트 비헤이비어 취소

기본 비헤이비어 취소와 관련된 두 메서드는 preventDefault() 및 isDefaultPrevented() 메서드입니다. 이벤트와 연결된 기본 비헤이비어를 취소하려면 preventDefault() 메서드를 호출합니다. 이벤트 객체에서 preventDefault() 메서드가 이미 호출되었는지 여부를 확인하려면 isDefaultPrevented() 메서드를 호출합니다. 이때 메서드가 이미 호출되었으면 true를 반환하고 그렇지 않으면 false를 반환합니다.

preventDefault() 메서드는 이벤트의 기본 비헤이비어를 취소할 수 있는 경우에만 작동합니다. 이를 확인하려면 API 설명서에서 해당 이벤트 유형에 대한 내용을 참조하거나, ActionScript를 사용하여 해당 이벤트 객체의 cancelable 속성을 검토하면 됩니다.

기본 비헤이비어를 취소해도 이벤트 흐름에서 이벤트 객체의 진행 상태에는 아무 영향을 주지 않습니다. 이벤트 흐름에서 이벤트 객체를 제거하려면 Event 클래스의 이벤트 흐름 메서드를 사용하십시오.

Event 클래스의 하위 클래스

대부분의 이벤트는 Event 클래스에 정의된 공통 속성 집합만 있어도 충분합니다. 그러나 Event 클래스에 있는 속성으로 캡처할 수 없는 고유한 특성을 갖고 있는 이벤트도 있습니다. 이러한 이벤트를 위해 Flash Player API에서는 Event 클래스의 하위 클래스를 정의합니다.

각 하위 클래스는 해당 이벤트 범주에 고유한 속성 및 이벤트 유형을 추가로 제공합니다. 예를 들어, 마우스 입력과 관련된 이벤트에는 Event 클래스에 정의된 속성으로 캡처할 수 없는 몇 가지 고유한 특성이 있습니다. MouseEvent 클래스는 마우스 이벤트의 위치, 마우스 이벤트를 실행하는 동안 특정 키를 눌렀는지 여부 등의 정보가 포함된 10개의 속성을 추가하여 Event 클래스를 확장합니다.

또한 Event 하위 클래스에는 하위 클래스와 관련된 이벤트 유형을 나타내는 상수도 포함되어 있습니다. 예를 들어, MouseEvent 클래스는 `click`, `doubleClick`, `mouseDown` 및 `mouseUp` 등 몇 가지 마우스 이벤트 유형에 대한 상수를 정의합니다.

306페이지의 “Event 클래스 유틸리티 메서드” 단원에서 설명한 것처럼 Event 하위 클래스를 만들 때는 `clone()` 및 `toString()` 메서드를 재정의하여 해당 하위 클래스에 고유한 기능을 제공해야 합니다.

이벤트 리스너

이벤트 리스너는 Flash Player에서 특정 이벤트에 대한 응답으로 실행되는 함수이며 이벤트 핸들러라고도 합니다. 이벤트 리스너를 추가할 때는 두 단계 과정을 거쳐야 하는데, 먼저 Flash Player에서 이벤트에 대한 응답으로 실행할 함수 또는 클래스 메서드를 만듭니다. 이를 리스너 함수 또는 이벤트 핸들러 함수라고도 합니다. 그런 다음 `addEventListener()` 메서드를 사용하여 리스너 함수를 이벤트 대상 또는 적절한 이벤트 흐름을 따라 놓여 있는 표시 목록 객체에 등록합니다.

리스너 함수 만들기

리스너 함수 만들기는 ActionScript 3.0 이벤트 모델이 DOM 이벤트 모델과 다른 부분 중 하나입니다. DOM 이벤트 모델에서는 이벤트 리스너와 리스너 함수 간의 차이가 명확합니다. 이벤트 리스너는 EventListener 인터페이스를 구현하는 클래스의 인스턴스이고 리스너 함수는 `handleEvent()`라는 클래스의 메서드입니다. 또한 DOM 이벤트 모델에서는 실제 리스너 함수가 아니라 리스너 함수를 포함하는 클래스 인스턴스를 등록합니다.

반면에 ActionScript 3.0 이벤트 모델에서는 이벤트 리스너와 리스너 함수 간에 차이가 없습니다. ActionScript 3.0에는 EventListener 인터페이스가 없으며 리스너 함수를 클래스 외부에 또는 클래스의 일부로 정의할 수 있습니다. 또한 리스너 함수의 이름을 `handleEvent()`로 지정할 필요가 없으며 유효한 식별자를 사용하여 명명할 수 있습니다. ActionScript 3.0에서는 실제 리스너 함수의 이름을 등록한다는 것도 DOM 이벤트 모델과 다릅니다.

클래스 외부에 정의되는 리스너 함수

다음 코드에서는 빨간색 정사각형 모양을 표시하는 간단한 SWF 파일을 만듭니다. 여기에서 클래스 외부에 정의된 `clickHandler()` 리스너 함수는 빨간색 정사각형에서 마우스 클릭 이벤트를 수신합니다.

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
```

```
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
}

function clickHandler(event:MouseEvent):void
{
  trace("clickHandler detected an event of type: " + event.type);
  trace("the this keyword refers to: " + this);
}
```

사용자가 사각형을 클릭하여 그 결과 나타나는 SWF 파일과 상호 작용하면 Flash Player에서 다음과 같은 추적 출력이 생성됩니다.

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

이벤트 객체는 `clickHandler()`에 인수로 전달됩니다. 이를 통해 리스너 함수에서 이벤트 객체를 검토할 수 있습니다. 이 예제에서는 이벤트 객체의 `type` 속성을 사용하여 이벤트가 click 이벤트인지 확인합니다.

또한 이 예제에서는 `this` 키워드의 값도 확인합니다. 이 경우에는 함수가 사용자 정의 클래스 또는 객체의 외부에 정의되므로 `this`가 전역 객체를 나타냅니다.

클래스 메서드로 정의되는 리스너 함수

다음 예제는 `clickHandler()` 함수가 ChildSprite 클래스의 메서드로 정의된다는 것만 제외하면 ClickExample 클래스를 정의하는 위의 예제와 같습니다.

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
  private function clickHandler(event:MouseEvent):void
  {
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
  }
}
```

사용자가 빨간색 사각형을 클릭하여 그 결과로 나타나는 SWF 파일에 대해 작업을 처리하면 Flash Player에서 다음과 같은 추적 출력이 생성됩니다.

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

여기에서 `this` 키워드는 `child`라는 ChildSprite 인스턴스를 참조하는데, 이는 ActionScript 2.0의 비헤이비어와 비교하여 달라진 점입니다. ActionScript 2.0 구성 요소를 사용한 경우, 클래스 메서드를 `UIEventDispatcher.addEventListener()`에 전달하면 메서드 범위가 리스너 메서드가 정의된 클래스 대신 이벤트를 브로드캐스팅하는 구성 요소에 바인딩되었음을 유념해야 합니다. 즉, ActionScript 2.0에서 이 방법을 사용한 경우에는 `this` 키워드가 ChildSprite 인스턴스 대신 이벤트를 브로드캐스팅하는 구성 요소를 참조합니다.

이렇게 되면 리스너 메서드를 포함하는 클래스의 다른 메서드 및 속성에 액세스할 수 없으므로 일부 프로그래머에게는 이것이 중요한 문제였고 이를 해결하기 위해 ActionScript 2.0 프로그래머는 mx.util.Delegate 클래스를 사용하여 리스너 메서드의 범위를 변경할 수 있었습니다. 그러나 ActionScript 3.0에서는 addEventListener()가 호출될 때 바인딩 메서드를 만들기 때문에 이 방법이 더 이상 필요하지 않습니다. 결과적으로 this 키워드는 child라는 ChildSprite 인스턴스를 참조하고 프로그래머는 ChildSprite 클래스의 다른 메서드와 속성에 액세스할 수 있습니다.

사용할 수 없는 이벤트 리스너

동적으로 지정된 리스너 함수를 가리키는 속성으로 일반 객체를 만드는 제 3의 방법이 있지만 사용하지 않는 것이 좋습니다. 이 방법은 ActionScript 2.0에서 일반적으로 사용되었기 때문에 여기에서 설명하지만 ActionScript 3.0에서는 사용하지 않아야 합니다. 이 방법을 사용하면 this 키워드가 리스너 객체 대신 전역 객체를 참조하므로, 권장할만한 방법은 아닙니다.

다음 예제는 리스너 함수가 myListenerObj라는 일반 객체의 일부로 정의된다는 것만 제외하면 위에 나오는 ClickExample 클래스 예제와 같습니다.

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
  }
}

var myListenerObj:Object = new Object();
```

```
myListenerObj.clickHandler = function (event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

추적 결과는 다음과 같습니다.

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

this 키워드가 myListenerObj를 참조하고 추적 출력이 [object Object]로 표시된다고 예상할 수 있지만 실제로는 전역 객체를 참조합니다. 동적 속성 이름을 addEventListener()에 인수로 전달하면 Flash Player에서 바인딩 메서드를 만들 수 없습니다. 이는 listener 매개 변수로 전달하는 대상이 리스너 함수의 메모리 주소이고 Flash Player에서는 해당 메모리 주소를 myListenerObj 인스턴스에 링크할 수 없기 때문입니다.

이벤트 리스너 관리

IEventDispatcher 인터페이스의 메서드를 사용하여 리스너 함수를 관리할 수 있습니다. IEventDispatcher 인터페이스는 DOM 이벤트 모델에서 사용되는 EventTarget 인터페이스의 ActionScript 3.0 버전입니다. IEventDispatcher라는 이름을 보면 이 인터페이스가 주로 이벤트 객체를 보내거나 전달하는 데 사용되는 것 같지만 실제로 이 클래스의 메서드는 이벤트 리스너를 등록, 확인 또는 제거하는 데 자주 사용됩니다. IEventDispatcher 인터페이스는 다음 코드와 같이 다섯 개의 메서드를 정의합니다.

```
package flash.events
{
  public interface IEventDispatcher
  {
    function addEventListener(eventName:String,
             listener:Object,
             useCapture:Boolean=false,
             priority:Integer=0,
             useWeakReference:Boolean=false):Boolean;

    function removeEventListener(eventName:String,
           listener:Object,
           useCapture:Boolean=false):Boolean;

    function dispatchEvent(eventObject:Event):Boolean;

    function hasEventListener(eventName:String):Boolean;
    function willTrigger(eventName:String):Boolean;
  }
}
```

Flash Player API는 이벤트 대상 또는 이벤트 흐름 구성 요소일 수 있는 모든 클래스의 기본 클래스로 사용되는 EventDispatcher 클래스를 사용하여 IEventDispatcher 인터페이스를 구현합니다. 예를 들어, DisplayObject 클래스는 EventDispatcher 클래스에서 상속됩니다. 즉, 표시 목록의 모든 객체는 IEventDispatcher 인터페이스의 메서드에 액세스할 수 있습니다.

이벤트 리스너 추가

`addEventListener()`는 IEventDispatcher 인터페이스에서 많은 역할을 담당하는 중요한 메서드입니다. 이 메서드는 리스너 함수를 등록할 때 사용됩니다. 이때 `type`과 `listener`라는 두 개의 매개 변수가 필요합니다. `type` 매개 변수는 이벤트 유형을 지정하는 데 사용되고, `listener` 매개 변수는 이벤트가 발생할 때 실행할 리스너 함수를 지정하는 데 사용됩니다. `listener` 매개 변수는 함수 또는 클래스 메서드에 대한 참조일 수 있습니다.

예외

`listener` 매개 변수를 지정할 때 괄호를 사용하지 마십시오. 예를 들어, 다음과 같이 `addEventListener()` 메서드를 호출하는 경우 `clickHandler()` 함수를 지정할 때 괄호를 추가하지 않습니다.

```
addEventListener(MouseEvent.CLICK, clickHandler).
```

`addEventListener()` 메서드의 `useCapture` 매개 변수를 사용하면 리스너를 활성화할 이벤트 흐름 단계를 제어할 수 있습니다. 예를 들어, `useCapture`가 `true`로 설정되면 이벤트 흐름의 캡처 단계를 진행하는 동안 리스너가 활성화되고, `useCapture`가 `false`로 설정되면 이벤트 흐름의 대상 단계 및 버블링 단계를 진행하는 동안 리스너가 활성화됩니다. 이벤트 흐름의 모든 단계를 진행하는 동안 이벤트를 수신하려면 `addEventListener()`를 두 번 호출해야 하는데, 첫 번째는 `useCapture`를 `true`로 설정하고 두 번째는 `useCapture`를 `false`로 설정해야 합니다.

`addEventListener()` 메서드의 `priority` 매개 변수는 DOM 레벨 3 이벤트 모델의 공식 구성 요소가 아닙니다. 이 매개 변수는 이벤트 리스너를 좀 더 융통성 있게 구성할 수 있도록 하기 위해 ActionScript 3.0에 포함되었습니다. `addEventListener()`를 호출하는 경우 정수 값을 `priority` 매개 변수로 전달하여 해당 이벤트 리스너의 우선 순위를 설정할 수 있습니다. 기본값은 0이지만 음의 정수 또는 양의 정수 값으로 설정할 수 있습니다. 이 값이 클수록 이벤트 리스너의 실행 순서가 빠릅니다. 우선 순위가 같은 이벤트 리스너는 추가된 순서대로 실행되므로 먼저 추가된 리스너의 실행 순서가 빠릅니다.

`useWeakReference` 매개 변수를 사용하면 리스너 함수에 대한 참조를 약한 참조 또는 일반 참조로 지정할 수 있습니다. 이 매개 변수를 `true`로 설정하면 더 이상 필요하지 않은 리스너 함수가 메모리에 남아있는 것을 방지할 수 있습니다. Flash Player에서는 *가비지 컬렉션*이라는 기법을 활용하여 더 이상 사용되지 않는 객체를 메모리에서 제거합니다. 객체에 대한 참조가 없는 경우 해당 객체는 더 이상 사용되지 않는 것으로 간주됩니다. 가비지 수집기는 참조가 약한 참조인지 여부에 관계없이 작업을 실행합니다. 즉, 약한 참조만 포함하는 리스너 함수를 가비지 컬렉션 대상으로 포함할 수 있습니다.

이 매개 변수를 사용하면 표시 객체의 이벤트로 작업할 수 있다는 사실도 중요합니다. 일반적으로 표시 객체가 표시 목록에서 제거되면 메모리에서도 제거된다고 예상할 것입니다. 그러나 useWeakReference 매개 변수를 false(기본값)로 설정하고 다른 객체가 표시 객체에 리스너로 등록한 경우에는 해당 표시 객체가 더 이상 화면에 나타나지 않지만 Flash Player의 메모리에는 그대로 유지됩니다. 이 문제를 해결하려면 useWeakReference 매개 변수를 true로 설정하고 모든 리스너를 표시 객체에 등록하거나, removeEventListener() 메서드를 사용하여 표시 객체에서 모든 이벤트 리스너를 제거하십시오.

이벤트 리스너 제거

removeEventListener() 메서드를 사용하면 더 이상 필요하지 않은 이벤트 리스너를 제거할 수 있습니다. 더 이상 사용하지 않을 모든 리스너를 제거하는 것이 좋습니다. 이 메서드를 사용하는 경우에는 addEventListener() 메서드를 사용할 때와 마찬가지로 eventName 및 listener 매개 변수가 반드시 필요합니다. 앞에서 설명했듯이 모든 이벤트 단계를 진행하는 동안 이벤트를 수신하려면 addEventListener()를 두 번 호출해야 하는데, 첫 번째는 useCapture를 true로 설정하고 두 번째는 false로 설정해야 합니다. 마찬가지로 두 이벤트 리스너를 모두 제거하려면 removeEventListener()를 두 번 호출해야 하는데, 첫 번째는 useCapture를 true로 설정하고 두 번째는 false로 설정해야 합니다.

이벤트 전달

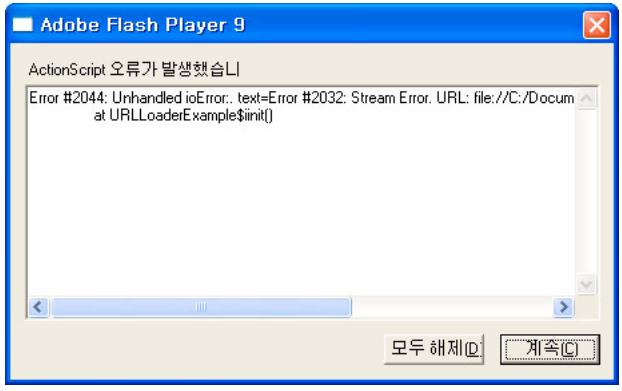
dispatchEvent() 메서드는 고급 프로그래머가 사용자 정의 이벤트 객체를 이벤트 흐름에 전달할 때 사용할 수 있습니다. 이 메서드에 사용할 수 있는 매개 변수는 이벤트 객체에 대한 참조뿐이며, 이는 Event 클래스의 인스턴스 또는 Event 클래스의 하위 클래스여야 합니다. 이벤트가 전달된 후 해당 이벤트 객체의 target 속성은 dispatchEvent()가 호출된 객체로 설정됩니다.

기존 이벤트 리스너 확인

IEventDispatcher 인터페이스의 마지막 두 메서드는 이벤트 리스너가 있는지 여부에 대해 유용한 정보를 제공합니다. 지정된 이벤트 유형의 이벤트 리스너가 특정 표시 목록 객체에 있는 경우 hasEventListener() 메서드에서 true를 반환합니다. 특정 표시 목록 객체에 리스너가 있는 경우 willTrigger() 메서드에서도 true를 반환하지만 willTrigger() 메서드는 해당 표시 객체뿐만 아니라 이벤트 흐름의 모든 단계에 대해 해당 표시 목록 객체의 모든 조상에서도 리스너를 확인합니다.

리스너가 없는 오류 이벤트

예외(이벤트 아님)는 ActionScript 3.0에서 오류 처리를 위한 주요 메커니즘으로 사용되지만 파일 로드와 같은 비동기 작업에는 예외 처리가 작동하지 않습니다. 이러한 비동기 작업을 실행하는 동안 오류가 발생하면 Flash Player에서는 오류 이벤트 객체를 전달합니다. 오류 이벤트에 대한 리스너를 만들지 않은 경우에는 Flash Player의 디버거 버전에서 오류 정보가 포함된 대화 상자가 표시됩니다. 예를 들어, 파일을 로드할 때 잘못된 URL을 사용하면 Flash Player의 디버거 버전에 다음과 같은 대화 상자가 표시됩니다.



대부분의 오류 이벤트는 ErrorEvent 클래스를 기초로 하며 Flash Player에 표시되는 오류 메시지를 저장하는 데 사용되는 `text` 속성을 포함합니다. 이때 StatusEvent 클래스와 NetStatusEvent 클래스는 예외입니다. 이러한 두 클래스에는 모두 `level` 속성(`StatusEvent.level` 및 `NetStatusEvent.info.level`)이 있으며, `level` 속성의 값이 `"error"`이면 이러한 이벤트 유형이 오류 이벤트로 간주됩니다.

오류 이벤트가 발생해도 SWF 파일은 계속 실행됩니다. 이러한 오류 이벤트는 디버거 버전의 브라우저 플러그인과 독립 실행형 플레이어의 대화 상자, 제작 플레이어 출력 패널의 메시지, Adobe Flex Builder 2 로그 파일의 항목 등으로만 표시되며 릴리스 버전의 Flash Player에는 표시되지 않습니다.

예제: 알람 시계

알람 시계 예제는 사용자가 알람이 울리는 시간 및 알람이 울릴 때 표시되는 메시지를 지정하는 데 사용할 수 있는 시계로 구성됩니다. 이 예제는 제5장, “날짜 및 시간을 사용한 작업”의 SimpleClock 응용 프로그램을 기초로 구성되며, 다음을 포함하여 ActionScript 3.0에서 이벤트를 사용하는 몇 가지 작업을 보여 줍니다.

- 이벤트 수신 및 응답
- 리스너에 이벤트 알림
- 사용자 정의 이벤트 유형 만들기

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. Alarm Clock 응용 프로그램 파일은 Samples/AlarmClock 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일을 포함합니다.

파일	설명
AlarmClockApp.mxml 또는 AlarmClockApp.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/clock/AlarmClock.as	알람 시계 기능을 추가하여 SimpleClock 클래스를 확장하는 클래스입니다.
com/example/programmingas3/clock/AlarmEvent.as	AlarmClock 클래스의 `alarm` 이벤트에 대한 이벤트 객체로 사용되는 사용자 정의 이벤트 클래스(flash.events.Event의 하위 클래스)입니다.
com/example/programmingas3/clock/AnalogClockFace.as	현재 시간을 기준으로 시계의 둥근 문자반과 시침, 분침, 초침을 그립니다(SimpleClock 예제의 설명 참조).
com/example/programmingas3/clock/SimpleClock.as	간단한 시간 계측 기능을 포함하는 시계 인터페이스 구성 요소입니다(SimpleClock 예제의 설명 참조).

알람 시계 개요

이 예제에 나오는 시계의 주요 기능(예: 시간 추적 및 시계 문자반 표시)은 189페이지의 “예제: 간단한 아날로그 시계”에서 설명하는 SimpleClock 응용 프로그램 코드를 다시 사용합니다. AlarmClock 클래스는 SimpleClock 예제에서 알람 시간 설정, 알람이 “울릴” 때 알림 제공 등 알람 시계에 필요한 기능을 추가하여 SimpleClock 클래스를 확장합니다.

특정 동작이 발생할 때 알림을 제공하는 것은 이벤트의 역할입니다. AlarmClock 클래스는 원하는 액션을 실행하기 위해 다른 객체에서 수신할 수 있는 Alarm 이벤트를 노출합니다. 또한 AlarmClock 클래스는 Timer 클래스의 인스턴스를 사용하여 알람 트리거 시기를 결정할 수 있습니다. Timer 클래스는 AlarmClock 클래스와 같이 지정된 시간이 경과하면 다른 객체(이 예제의 경우 AlarmClock 인스턴스)에 알리는 이벤트를 제공합니다. 대부분의 ActionScript 응용 프로그램과 마찬가지로 이벤트는 Alarm Clock 샘플 응용 프로그램 기능의 중요한 부분입니다.

알람 트리거

앞에서 설명했듯이 AlarmClock 클래스에서 실제로 제공하는 기능은 알람 설정 및 트리거뿐입니다. 개발자는 내장 Timer 클래스(flash.utils.Timer)를 통해 지정된 시간이 경과한 후 실행할 코드를 정의할 수 있습니다. AlarmClock 클래스는 Timer 인스턴스를 사용하여 알람 실행 시기를 결정합니다.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * 알람에 사용할 Timer
 */
public var alarmTimer:Timer;
...
/**
 * 지정된 크기의 새 AlarmClock 을 인스턴스화합니다 .
 */
public override function initClock(faceSize:Number = 200):void
{
  super.initClock(faceSize);
  alarmTimer = new Timer(0, 1);
  alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

AlarmClock 클래스에 정의된 Timer 인스턴스 이름은 alarmTimer입니다. initClock() 메서드는 AlarmClock 인스턴스에 필요한 설정 작업을 수행하며 alarmTimer 변수를 사용하여 두 가지 작업을 실행합니다. 먼저 Timer 인스턴스에서 0밀리초 동안 대기한 후 타이머 이벤트를 한 번 트리거하도록 지시하는 매개 변수를 사용하여 변수가 인스턴스화되고, alarmTimer 변수를 인스턴스화한 후에 이 변수의 addEventListener() 메서드를 호출하여 변수의 timer 이벤트를 수신 대기하도록 요청합니다. 지정된 시간이 경과한 후 timer 이벤트를 전달하면 Timer 인스턴스가 작동합니다. 알람을 울리려면 AlarmClock 클래스에서 timer 이벤트 전달 시기를 알고 있어야 합니다. addEventListener()를 호출하면 AlarmClock 코드가 alarmTimer에 리스너로 등록됩니다. 두 매개 변수는 AlarmClock 클래스에서 timer 이벤트(TimerEvent.TIMER 상수로 나타냄)를 수신 대기할 수 있도록 요청하며, 이벤트가 발생하면 해당 이벤트에 대한 응답으로 AlarmClock 클래스의 onAlarm() 메서드를 호출해야 함을 나타냅니다.

실제로 알람을 설정하기 위해 다음과 같이 AlarmClock 클래스의 setAlarm() 메서드가 호출됩니다.

```
/**
 * 알람이 울리는 시간을 설정합니다 .
 * @param hour 알람 시간의 시 부분
 * @param minutes 알람 시간의 분 부분
 * @param message 알람이 울릴 때 표시할 메시지
 * @return 알람이 울리는 시간
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0,
message:String = "Alarm!"):Date
{
  this.alarmMessage = message;
  var now:Date = new Date();
  // 오늘 날짜의 이 시간을 만듭니다 .
  alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

  // 지정한 시간이 이미 오늘을 지났는지 확인합니다 .
  if (alarmTime <= now)
  {
    alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
  }

  // 알람 타이머가 설정되어 있는 경우 알람 타이머를 중지합니다 .
  alarmTimer.reset();
  // 알람이 울릴 때까지 남은 시간 ( 알람 시간과 현재 시간 간의 차이 ) 을
  // 밀리초 단위로 계산하고 해당 값을 알람 타이머의 delay 속성 값으로
  // 설정합니다 .
  alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
  alarmTimer.start();

  return alarmTime;
}
```

이 메서드는 알람 메시지를 저장하고, 실제로 알람이 울리는 시기를 나타내는 Date 객체(alarmTime)를 만드는 등 여러 가지 작업을 실행합니다. 메서드 정의의 마지막 몇 행에서는 현재 문서에서 주로 다루었던 내용인 alarmTimer 변수의 타이머를 설정하고 활성화합니다.

먼저 reset() 메서드가 호출되어 이미 실행되고 있는 타이머를 중단하고 재설정합니다. 그런 다음, alarmTime 변수의 값에서 현재 시간(now 변수로 나타냄)을 빼 알람이 울리기 전에 경과해야 할 시간(밀리초)을 확인합니다. Timer 클래스는 절대 시간으로 timer 이벤트를 트리거하지 않으므로 alarmTimer의 delay 속성에는 이러한 상대 시간 차이가 지정됩니다.

마지막으로 start() 메서드를 호출하여 실제로 타이머를 시작합니다.

지정된 시간이 경과하면 alarmTimer에서 timer 이벤트를 전달합니다. AlarmClock 클래스는 onAlarm() 메서드를 해당 이벤트의 리스너로 등록했으므로 timer 이벤트가 발생하면 onAlarm()이 호출됩니다.

```
/**
 * 타이머 이벤트 전달 시 호출됩니다 .
 */
public function onAlarm(event:TimerEvent):void
{
  trace("Alarm!");
  var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
  this.dispatchEvent(alarm);
}
```

이벤트 리스너로 등록되는 메서드는 적절한 서명, 즉 메서드의 매개 변수 집합 및 반환 유형을 사용하여 정의해야 합니다. 예를 들어, Timer 클래스의 timer 이벤트에 대한 리스너로 등록할 메서드는 데이터 유형이 Event 클래스의 하위 클래스인 TimerEvent(flash.events.TimerEvent)로 설정된 하나의 매개 변수를 정의해야 합니다. 그러면 Timer 인스턴스에서 이벤트 리스너를 호출하는 경우 TimerEvent 인스턴스가 이벤트 객체로 전달됩니다.

다른 코드에 알람 알림

AlarmClock 클래스는 Timer 클래스와 마찬가지로 알람이 울릴 때 다른 코드에서 알림을 받을 수 있도록 이벤트를 제공합니다. 클래스에서 ActionScript에 내장된 이벤트 처리 프레임워크를 사용하려면 해당 클래스에서 flash.events.IEventDispatcher 인터페이스를 구현해야 합니다. 일반적으로 이 작업을 수행하려면 flash.events.EventDispatcher 클래스를 확장하여 IEventDispatcher를 구현하거나, EventDispatcher의 하위 클래스 중 하나를 확장합니다. 앞에서 설명했듯이 AlarmClock 클래스는 SimpleClock 클래스를 확장하여 Sprite 클래스를 확장하고 결국 상속 체인을 통해 EventDispatcher 클래스를 확장합니다. 즉, AlarmClock 클래스에는 고유한 이벤트를 제공하는 기능이 이미 포함되어 있습니다.

EventDispatcher에서 AlarmClock에 상속되는 addEventListener() 메서드를 호출하면 다른 코드에 AlarmClock 클래스의 alarm 이벤트를 알리도록 등록할 수 있습니다. AlarmClock 인스턴스에서 alarm 이벤트가 발생했음을 다른 코드에 알릴 준비가 되면 EventDispatcher에서 상속되는 dispatchEvent() 메서드를 호출합니다.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

이러한 코드 행은 AlarmClock 클래스의 onAlarm() 메서드에서 가져온 것입니다(전체 코드는 앞에 있음). AlarmClock 인스턴스의 dispatchEvent() 메서드가 호출되면 등록된 모든 리스너에 AlarmClock 인스턴스의 alarm 이벤트가 트리거되었음을 알립니다. dispatchEvent()에 전달되는 매개 변수는 리스너 메서드에 전달될 이벤트 객체이며, 여기에서는 AlarmEvent 클래스의 인스턴스입니다. AlarmEvent 클래스는 이 예제를 위해 만든 Event 하위 클래스입니다.

사용자 정의 alarm 이벤트 제공

모든 이벤트 리스너는 트리거되는 특정 이벤트에 대한 정보가 포함된 이벤트 객체 매개 변수를 받습니다. 대부분의 경우 이벤트 객체는 Event 클래스의 인스턴스입니다. 그러나 경우에 따라서는 이벤트 리스너에 추가 정보를 제공하면 유용합니다. 앞에서 설명했듯이 일반적으로 이렇게 하려면 Event 클래스의 하위 클래스인 새 클래스를 정의하고 해당 클래스의 인스턴스를 이벤트 객체로 사용하면 됩니다. 이 예제에서는 AlarmClock 클래스의 alarm 이벤트가 전달되면 AlarmEvent 인스턴스가 이벤트 객체로 사용됩니다. 여기에 표시된 AlarmEvent 클래스는 alarm 이벤트, 특히 알람 메시지에 대한 추가 정보를 제공합니다.

```
import flash.events.Event;

/**
 * 이 사용자 정의 Event 클래스는 기본 Event에 message 속성을 추가합니다.
 */
public class AlarmEvent extends Event
{
    /**
     * 새로운 AlarmEvent 유형의 이름
     */
    public static const ALARM:String = "alarm";

    /**
     * 이 이벤트 객체를 가진 이벤트 핸들러에 전달할 수 있는
     * 텍스트 메시지
     */
    public var message:String;

    /**
     *  생성자
     * @param message 알람이 울릴 때 표시할 텍스트
```

```
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
      super(ALARM);
      this.message = message;
    }
    ...
}
```

사용자 정의 이벤트 객체 클래스를 만들려면 위의 예제와 같이 Event 클래스를 확장하는 클래스를 정의하는 것이 가장 좋습니다. 상속된 기능을 보완하기 위해 AlarmEvent 클래스는 이벤트와 관련된 알람 메시지의 텍스트를 포함하는 `message` 속성을 정의합니다. 이 `message` 값은 AlarmEvent 생성자의 매개 변수로 전달됩니다. 또한 AlarmEvent 클래스는 AlarmClock 클래스의 `addEventListener()` 메서드를 호출할 때 특정 이벤트(`alarm`)를 참조하는 데 사용할 수 있는 `ALARM` 상수도 정의합니다.

모든 Event 하위 클래스는 사용자 정의 기능을 추가해야 할 뿐만 아니라 ActionScript 이벤트 처리 프레임워크의 일부로 상속 `clone()` 메서드를 재정의해야 합니다. 또한 Event 하위 클래스는 상속된 `toString()` 메서드를 재정의하여 `toString()` 메서드 호출의 반환값에 사용자 정의 이벤트의 속성이 포함되도록 할 수도 있습니다.

```
/**
 * 현재 인스턴스의 복사본을 만들어 반환합니다 .
 * @return 현재 인스턴스의 복사본
 */
public override function clone():Event
{
  return new AlarmEvent(message);
}

/**
 * 현재 인스턴스의 모든 속성을 포함하는 String 을
 * 반환합니다 .
 * @return 현재 인스턴스의 문자열 표현
 */
public override function toString():String
{
  return formatToString("AlarmEvent", "type", "bubbles", "cancelable",
"eventPhase", "message");
}
```

재정의된 `clone()` 메서드는 모든 사용자 정의 속성이 현재 인스턴스와 일치하도록 설정된 사용자 정의 Event 하위 클래스의 새 인스턴스를 반환해야 합니다. 재정의된 `toString()` 메서드에서 유틸리티 메서드 `formatToString()`(Event에서 상속됨)은 사용자 정의 유형의 이름 및 모든 속성의 이름과 값으로 이루어진 문자열을 제공하는 데 사용됩니다.

제 11 장

11 XML을 사용한 작업

ActionScript 3.0에는 E4X(ECMAScript for XML) 사양인 ECMA-357 버전 2에 기초한 클래스 그룹이 포함되어 있습니다. 이러한 클래스에서는 XML 데이터로 작업할 수 있도록 강력하고 사용하기 쉬운 기능을 제공합니다. E4X를 사용하면 이전의 프로그래밍 기법보다 빠르게 XML 데이터로 코드를 개발할 수 있습니다. 또한 코드를 쉽게 읽을 수 있다는 이점도 있습니다.

이 장에서는 E4X를 사용하여 XML 데이터를 처리하는 방법에 대해 설명합니다.

목차

XML의 기초

XML을 사용한 작업 소개

XML은 구조화된 정보를 표시하는 표준 방법으로, 컴퓨터에서 처리가 간편하고 사용자 역시 비교적 쉽게 작성하고 이해할 수 있습니다. XML은 eXtensible Markup Language의 약어이며, XML 표준은 www.w3.org/XML/에서 사용할 수 있습니다.

XML은 편리하게 데이터를 범주화할 수 있는 표준 방식을 통해 데이터에 대한 읽기, 액세스 및 조작 작업을 용이하게 합니다. XML에서는 HTML에서와 유사한 트리 구조 및 태그 구조를 사용합니다. 다음은 간단한 XML 데이터의 예입니다.

```
<song>
  <title>What you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML 데이터는 속성 및 기타 구조적 구성 요소뿐만 아니라 다른 태그에 중첩된 태그가 있는 더 복잡한 형태일 수 있습니다. 다음은 복잡한 XML 데이터의 예입니다.

```
<album>
  <title>Questions, unanswered</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <tracks>
    <song tracknumber="1" length="4:05">
      <title>What do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:31</lastplayed>
    </song>
    <song tracknumber="2" length="3:45">
      <title>Who do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:35</lastplayed>
    </song>
    <song tracknumber="3" length="5:14">
      <title>When do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:39</lastplayed>
    </song>
    <song tracknumber="4" length="4:19">
      <title>Do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:44</lastplayed>
    </song>
```

```
  </tracks>
</album>
```

이 XML 문서에는 다른 완전한 XML 구조(예: 자식이 있는 `song` 태그)가 들어 있습니다. 또한 특성(예: `song` 태그의 `tracknumber` 및 `length`)과 같은 다른 XML 구조를 비롯하여 데이터가 아닌 태그로 구성된 태그(예: `tracks` 태그)를 확인할 수 있습니다.

XML 시작

XML 관련 지식이 전무하거나 거의 없다면 여기에서 소개하는 XML 데이터의 가장 일반적인 사항에 대한 설명을 참조하십시오. XML 데이터는 일반 텍스트 형식으로 작성되며, 정보를 구조화된 형식으로 구성하는 특수 구문이 사용됩니다. 일반적으로 하나의 XML 데이터 집합을 *XML 문서*라고 합니다. XML 형식에서 데이터는 계층 구조에 따라 *요소*로 구성됩니다. 요소는 하나의 데이터 항목이거나 다른 여러 요소의 컨테이너일 수 있습니다. 모든 XML 문서는 최상위 수준(기본 항목)에 하나의 요소를 갖고 있습니다. 대부분 여러 중첩 요소로 구성되는 경우가 많지만 이 루트 요소 내에는 단일 정보가 포함됩니다. 한 예로, 음악 앨범 정보가 포함된 다음 XML 문서를 살펴보겠습니다.

```
<song tracknumber="1" length="4:05">
  <title>What do you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <mood>Happy</mood>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

각 요소는 여러 *태그*, 즉 각괄호(보다 작음 및 보다 큼 기호)로 둘러싸인 요소 이름으로 구분됩니다. 요소의 시작을 나타내는 여는 태그의 이름은 다음과 같습니다.

```
<title>
```

요소의 끝을 나타내는 닫는 태그에는 다음과 같이 요소 이름 앞에 슬래시가 사용됩니다.

```
</title>
```

요소에 내용이 없는 경우에는 빈 요소(자체 닫기 요소라고도 함)로 작성될 수 있습니다. XML에서

```
<lastplayed/>
```

다음 요소와 동일합니다.

```
<lastplayed></lastplayed>
```

여는 태그와 닫는 태그 사이에 포함된 내용 외에도 요소에는 여는 태그로 정의된 *특성*이라는 다른 값이 포함될 수 있습니다. 예를 들어, 다음 XML 요소는 값이 "`4:19`"인 한 개의 특성(`length`)을 정의합니다.

```
<song length="4:19"></song>
```

각 XML 요소에는 단일 값 또는 하나 이상의 XML 요소가 있거나 빈 요소의 경우 내용이 없습니다.

XML에 대한 세부 정보

XML에 대해 보다 자세한 내용을 원하는 경우, 다음 웹 사이트를 비롯하여 다양한 서적 및 리소스를 활용할 수 있습니다.

- W3Schools XML 자습서: http://w3schools.com/xml/
- XML.com: http://www.xml.com/
- XMLpitstop 자습서, 토론 목록 및 기타: http://xmlpitstop.com/

XML 작업을 위한 ActionScript 클래스

ActionScript 3.0에는 XML 구조의 정보로 작업할 때 사용할 수 있는 여러 클래스가 포함되어 있습니다. 그 중 두 가지 주요 클래스는 다음과 같습니다.

- XML: 하나의 XML 요소를 나타내며, 복수의 자식 또는 단일 값 요소로 구성된 XML 문서일 수 있습니다.
- XMLList: 일련의 XML 요소를 나타냅니다. XMLList 객체는 "형제" XML 요소가 여럿일 경우 사용됩니다. 형제 XML 요소란 XML 문서의 계층 내에서 부모 및 수준이 동일한 요소를 의미합니다. 예를 들어, 다음과 같은 일련의 XML 요소(XML 문서에 포함된 요소라고 가정)와 관련된 작업에서는 XMLList 인스턴스를 사용하는 것이 가장 간편할 수 있습니다.

```
<artist type="composer">Fred Wilson</artist>
<artist type="conductor">James Schmidt</artist>
<artist type="soloist">Susan Harriet Thurndon</artist>
```

ActionScript에 포함된 Namespace 및 QName 클래스를 사용하면 XML 네임스페이스와 관련된 고급 기능을 활용할 수 있습니다. 자세한 내용은 342페이지의 "XML 네임스페이스 사용"을 참조하십시오.

XML 작업을 지원하기 위한 내장 클래스 이외에도 ActionScript 3.0에는 XML 데이터 액세스 및 조작을 위한 특수 기능을 제공하는 여러 연산자가 포함되어 있습니다. 이러한 클래스 및 연산자를 사용한 XML 작업 방법은 E4X(ECMAScript for XML)라고 하며 ECMA-357 Edition 2 사양에 정의되어 있습니다.

일반적인 XML 작업

ActionScript에서 XML을 사용할 경우 다음과 같은 작업을 수행하게 됩니다.

- XML 문서 구성(요소 및 값 추가)
- XML 요소, 값 및 특성 액세스
- XML 요소 필터링(검색)
- XML 요소 집합 반복
- XML 클래스와 String 클래스 간 데이터 변환
- XML 네임스페이스를 사용한 작업
- 외부 XML 파일 로드

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- 요소: XML 문서에 있는 단일 항목으로, 시작 태그 및 끝 태그 사이의 내용 및 태그로 식별됩니다. XML 요소는 텍스트 데이터 또는 기타 요소로 구성되거나 비어 있을 수 있습니다.
- 비어 있는 요소: 자식 요소가 없는 XML 요소입니다. 비어 있는 요소는 자체 닫는 태그(예: `<element/>`)라고도 합니다.
- 문서: 하나의 XML 구조를 나타냅니다. XML 문서는 여러 개의 요소 또는 비어 있는 하나의 요소로 구성될 수 있지만, 문서의 기타 모든 요소를 포함하는 최상위 수준 요소가 반드시 하나씩 있어야 합니다.
- 노드: XML 요소를 지칭하는 다른 이름입니다.
- 특성: 요소와 관련이 있으며 이름을 갖는 값으로, 요소 내에 중첩된 별도의 자식 요소로 작성되지 않고 요소의 여는 태그에 `attributename="value"` 형식과 같이 작성됩니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 기본적으로 이 장의 모든 코드 샘플에는 해당하는 `trace()` 함수 호출이 포함되어 있습니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   [출력] 패널에서 `trace()` 함수의 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 모든 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

E4X를 사용하여 XML 처리

ECMAScript for XML 사양은 XML 데이터 작업을 위한 클래스 및 기능 집합을 정의합니다. 이러한 클래스와 기능을 하나로 묶어 *E4X*라고 합니다. ActionScript 3.0에는 XML, XMLList, QName 및 Namespace와 같은 E4X 클래스가 포함되어 있습니다.

E4X 클래스의 메서드, 속성 및 연산자는 다음을 구현할 수 있도록 디자인되었습니다.

- 단순성 - 가능한 경우 E4X를 사용하면 XML 데이터 작업을 위한 코드를 쉽게 작성하고 이해할 수 있습니다.
- 일관성 - E4X에 내포되어 있는 이론 및 메서드는 내부적으로 일관되고 ActionScript의 다른 요소와 일치합니다.
- 익숙함 - 도트(.) 연산자와 같이 잘 알려진 연산자를 사용하여 XML 데이터를 조작합니다.

참고

ActionScript 2.0에는 XML 클래스가 있습니다. ActionScript 3.0에서는 이 클래스가 E4X에 포함된 ActionScript 3.0 XML 클래스와 충돌하지 않도록 XMLDocument로 이름이 바뀌었습니다. ActionScript 3.0에서는 이전 클래스, 즉 XMLDocument, XMLNode, XMLParser 및 XMLTag가 이전 버전을 지원하기 위해 flash.xml 패키지에 포함되어 있습니다. 새로 추가된 E4X 클래스는 기본 클래스이므로 사용할 때 패키지를 가져올 필요가 없습니다. 이 장에서는 이전 ActionScript 2.0 XML 클래스에 대해 자세히 설명하지 않습니다. 이러한 클래스에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 flash.xml 패키지를 참조하십시오.

다음은 E4X를 사용하여 데이터를 조작하는 예제입니다.

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

대개 응용 프로그램에서는 웹 서비스 또는 RSS 피드와 같은 외부 소스에서 XML 데이터를 로드합니다. 그러나 이 장의 예제에서는 사용자가 쉽게 이해할 수 있도록 XML 데이터를 리터럴로 지정합니다.

다음 코드와 같이 E4X에는 도트(.) 및 특성 식별자(@) 연산자처럼 XML에서 속성과 특성에 액세스하는 데 사용할 수 있도록 직관적인 연산자가 포함되어 있습니다.

```
trace(myXML.item[0].menuName); // 출력 : burger
trace(myXML.item.(@id==2).menuName); // 출력 : fries
trace(myXML.item.(menuName=="burger").price); // 출력 : 3.95
```

다음 코드 예제와 같이 appendChild() 메서드를 사용하면 새 자식 노드를 XML에 지정할 수 있습니다.

```
var newItem:XML =
  <item id="3">
    <menuName>medium cola</menuName>
    <price>1.25</price>
  </item>

myXML.appendChild(newItem);
```

다음과 같이 @ 및 . 연산자를 사용하면 데이터를 읽을 수 있을 뿐만 아니라 지정할 수도 있습니다.

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

다음과 같이 for 루프를 사용하면 XML의 노드를 반복할 수 있습니다.

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
  var q:int = Number(property.@quantity);
  var p:Number = Number(property.price);
  var itemTotal:Number = q * p;
  total += itemTotal;
  trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

XML 객체

XML 객체는 XML 요소, 특성, 주석, 처리 명령 또는 텍스트 요소를 나타낼 수 있습니다.

XML 객체는 *간단한 내용*을 포함하는 객체 또는 *복잡한 내용*을 포함하는 객체로 분류됩니다. 자식 노드가 있는 XML 객체는 복잡한 내용을 포함하는 객체로 분류되고 특성, 주석, 처리 명령 또는 텍스트 노드 중 하나에 해당하는 XML 객체는 간단한 내용을 포함하는 객체로 분류됩니다.

예를 들어, 다음 XML 객체에는 주석과 처리 명령 등의 복잡한 내용이 포함되어 있습니다.

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
  <order>
    <!--This is a comment. -->
    <?PROC_INSTR sample ?>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

이제 다음 예제와 같이 `comments()` 및 `processingInstructions()` 메서드를 사용하여 새 XML 객체, 즉 주석과 처리 명령을 만들 수 있습니다.

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML 속성

XML 클래스에는 다음과 같은 다섯 개의 정적 속성이 포함되어 있습니다.

- `ignoreComments` 및 `ignoreProcessingInstructions` 속성은 XML 객체를 파싱할 때 주석 또는 처리 명령을 무시할지 여부를 결정합니다.
- `ignoreWhitespace` 속성은 공백 문자로만 구분되는 포함된 표현식 및 요소 태그에서 공백 문자를 무시할지 여부를 결정합니다.
- `prettyIndent` 및 `prettyPrinting` 속성은 XML 클래스의 `toString()` 및 `toXMLString()` 메서드에서 반환하는 텍스트의 서식을 지정하는 데 사용됩니다.

이러한 속성에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

XML 메서드

다음 메서드를 사용하면 계층 구조 형식의 XML 객체로 작업할 수 있습니다.

- appendChild()
- child()
- childIndex()
- children()
- descendants()
- elements()
- insertChildAfter()
- insertChildBefore()
- parent()
- prependChild()

다음 메서드를 사용하면 XML 객체 특성으로 작업할 수 있습니다.

- attribute()
- attributes()

다음 메서드를 사용하면 XML 객체 속성으로 작업할 수 있습니다.

- hasOwnProperty()
- propertyIsEnumerable()
- replace()
- setChildren()

다음 메서드를 사용하면 정규화된 이름 및 네임스페이스로 작업할 수 있습니다.

- addNamespace()
- inScopeNamespaces()
- localName()
- name()
- namespace()
- namespaceDeclarations()
- removeNamespace()
- setLocalName()
- setName()
- setNamespace()

다음 메서드를 사용하면 특정 유형의 XML 내용을 결정하고 작업할 수 있습니다.

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

다음 메서드를 사용하면 XML 객체를 문자열로 변환하고 서식을 지정할 수 있습니다.

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

다음과 같은 메서드도 추가로 사용할 수 있습니다.

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

이러한 메서드에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

XMLList 객체

XMLList 인스턴스는 임의의 XML 객체 컬렉션을 나타냅니다. 여기에는 전체 XML 문서, XML 조각 또는 XML 쿼리 결과 등이 포함될 수 있습니다.

다음 메서드를 사용하면 계층 구조 형식의 XMLList 객체로 작업할 수 있습니다.

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

다음 메서드를 사용하면 XMLList 객체 특성으로 작업할 수 있습니다.

- `attribute()`
- `attributes()`

다음 메서드를 사용하면 XMLList 속성으로 작업할 수 있습니다.

- `hasOwnProperty()`
- `propertyIsEnumerable()`

다음 메서드를 사용하면 특정 유형의 XML 내용을 결정하고 작업할 수 있습니다.

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

다음 메서드를 사용하면 XMLList 객체를 문자열로 변환하고 서식을 지정할 수 있습니다.

- `normalize()`
- `toString()`
- `toXMLString()`

다음과 같은 메서드도 추가로 사용할 수 있습니다.

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

이러한 메서드에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

XML 요소가 하나만 포함된 XMLList 객체는 XML 객체와 같은 것으로 간주되므로 XML 클래스의 모든 속성과 메서드를 사용할 수 있습니다. 예를 들어, 다음 코드에서 `doc.div`는 요소가 하나만 포함된 XMLList 객체이므로 XML 클래스에서 `appendChild()` 메서드를 사용할 수 있습니다.

```
var doc:XML =
    <body>
      <div>
        <p>Hello</p>
      </div>
    </body>;
doc.div.appendChild(<p>World</p>);
```

XML 속성 및 메서드 목록을 보려면 330페이지의 “XML 객체”를 참조하십시오.

XML 변수 초기화

다음과 같이 XML 객체에 XML 리터럴을 지정할 수 있습니다.

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

다음 코드 예제와 같이 `new` 생성자를 사용하여 XML 데이터가 포함된 문자열에서 XML 객체의 인스턴스를 만들 수도 있습니다.

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

닫는 태그가 없는 것과 같이 문자열의 XML 데이터 형식이 잘못된 경우에는 런타임 오류가 발생합니다.

다음 예제와 같이 다른 변수의 데이터를 참조로 XML 객체에 전달할 수도 있습니다.

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = “5”;
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</
  {tagname}>;
trace(x.toXMLString())
  // 출력 : <item id="5">Chicken</item>
```

URL에서 XML 데이터를 로드하려면 다음 예제와 같이 URLLoader 클래스를 사용합니다.

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
  var loader:URLLoader = event.target as URLLoader;
  if (loader != null)
  {
    externalXML = new XML(loader.data);
    trace(externalXML.toXMLString());
  }
  else
  {
    trace("loader is not a URLLoader!");
  }
}
```

소켓 연결에서 XML 데이터를 읽으려면 XMLSocket 클래스를 사용합니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 XMLSocket 항목을 참조하십시오.

XML 객체 어셈블 및 변환

XML 객체 속성 목록의 맨 앞이나 끝에 속성을 추가하려면 다음 예제와 같이 `prependChild()` 메서드 또는 `appendChild()` 메서드를 사용합니다.

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
  // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

지정된 속성의 앞이나 뒤에 속성을 추가하려면 다음과 같이 `insertChildBefore()` 메서드 또는 `insertChildAfter()` 메서드를 사용합니다.

```
var x:XML =
  <body>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
  </body>
```

```
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

다음 예제와 같이 XML 객체를 구성할 때 중괄호 연산자({ 및 })를 사용하여 다른 변수의 데이터를 참조로 전달할 수도 있습니다.

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
  var newnode:XML = new XML();
  newnode =
    <employee id={ids[i]}>
      <last>{names[i][0]}</last>
      <first>{names[i][1]}</first>
    </employee>;

  x = x.appendChild(newnode)
}
```

다음과 같이 = 연산자를 사용하여 속성 및 특성을 XML 객체에 지정할 수 있습니다.

```
var x:XML =
  <employee>
    <lastname>Smith</lastname>
  </employee>
x.firstname = "Jean";
x.@id = "239";
```

이렇게 하면 XML 객체 x가 다음과 같이 설정됩니다.

```
<employee id="239">
  <lastname>Smith</lastname>
  <firstname>Jean</firstname>
</employee>
```

+ 및 += 연산자를 사용하면 XMLList 객체를 연결할 수 있습니다.

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

이렇게 하면 XMLList 객체 xList가 다음과 같이 설정됩니다.

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

XML 구조 순회

XML의 강력한 기능 중 하나는 선형 텍스트 문자열을 통해 복잡한 중첩 데이터를 제공하는 것입니다. 데이터를 XML 객체로 로드하면 ActionScript에서 해당 데이터를 파싱하고 계층 구조를 메모리로 로드합니다. 이때 XML 데이터 형식이 잘못된 경우에는 런타임 오류가 발생합니다.

XML 및 XMLList 객체의 연산자와 메서드를 사용하면 XML 데이터 구조를 쉽게 순회할 수 있습니다.

도트(.) 연산자 및 자손 접근자(..) 연산자를 사용하면 XML 객체의 자식 속성에 액세스할 수 있습니다. 다음과 같은 XML 객체를 검토하십시오.

```
var myXML:XML =
  <order>
    <book ISBN="0942407296">
      <title>Baking Extravagant Pastries with Kumquats</title>
      <author>
        <lastName>Contino</lastName>
        <firstName>Chuck</firstName>
      </author>
      <pageCount>238</pageCount>
    </book>
    <book ISBN="0865436401">
      <title>Emu Care and Breeding</title>
      <editor>
        <lastName>Case</lastName>
        <firstName>Justin</firstName>
      </editor>
      <pageCount>115</pageCount>
    </book>
  </order>
```

`myXML.book` 객체는 `myXML` 객체의 자식 속성(`book`)을 포함하는 XMLList 객체입니다. 이러한 두 XML 객체는 `myXML` 객체의 두 `book` 속성에 대응됩니다.

`myXML..lastName` 객체는 이름이 `lastName`인 하위 속성을 포함하는 XMLList 객체입니다. 이러한 두 XML 객체는 `myXML` 객체의 두 `lastName` 속성에 대응됩니다.

`myXML.book.editor.lastName` 객체는 `lastName`이라는 자식 속성을 포함하는 XMLList 객체입니다. 여기에서 lastName은 `editor`의 자식 속성이며 editor는 `book`의 자식 속성입니다. 또한 book은 `myXML` 객체의 자식입니다. 이 예제의 경우에는 XML 객체(값이 `"Case"`로 설정된 `lastName` 속성)를 하나만 포함하는 XMLList 객체입니다.

부모 및 자식 노드 액세스

`parent()` 메서드는 XML 객체의 부모를 반환합니다.

자식 목록의 서수 인덱스 값을 사용하여 특정 자식 객체에 액세스할 수 있습니다. 예를 들어, `myXML`이라는 XML 객체에 이름이 `book`인 자식 속성이 두 개 포함되어 있는 경우를 가정해 봅니다. 이때 각 자식 속성 `book`에는 인덱스 번호가 연결되어 있습니다.

```
myXML.book[0]
myXML.book[1]
```

특정 손자 항목에 액세스하려면 자식 이름과 손자 이름에 모두 인덱스 번호를 지정해야 합니다.

```
myXML.book[0].title[0]
```

그러나 이름이 `title`인 `x.book[0]`의 자식이 하나뿐인 경우에는 다음과 같이 인덱스 참조를 생략할 수 있습니다.

```
myXML.book[0].title
```

마찬가지로 `x` 객체의 book 자식이 하나뿐이고 해당 자식 객체에 title 객체가 하나뿐인 경우에는 다음과 같이 두 인덱스 참조를 모두 생략할 수 있습니다.

```
myXML.book.title
```

다음 예제와 같이 `child()` 메서드를 사용하여 변수 또는 표현식을 기반으로 이름에 따라 자식 항목을 탐색할 수 있습니다.

```
var myXML:XML =
    <order>
      <book>
        <title>Dictionary</title>
      </book>
    </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // 출력 : Dictionary
```

특성 액세스

다음 코드와 같이 XML 또는 XMLList 객체에서 특성에 액세스하려면 @ 심볼(특성 식별자 연산자)을 사용합니다.

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@id); // 6401
```

다음 코드와 같이 * 와일드카드 심볼을 @ 심볼과 함께 사용하면 XML 또는 XMLList 객체의 모든 특성에 액세스할 수 있습니다.

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

다음 코드와 같이 `attribute()` 또는 `attributes()` 메서드를 사용하면 XML 또는 XMLList 객체의 특정 특성이나 모든 특성에 액세스할 수 있습니다.

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

다음 예제와 같이 아래 구문을 사용하여 특성에 액세스할 수도 있습니다.

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

이러한 구문은 각각 `employee.@id`와 동일하지만 `employee.@id` 구문을 사용하는 것이 좋습니다.

특성 또는 요소 값으로 필터링

괄호 연산자인 (및)를 사용하면 특정 요소 이름 또는 특성 값으로 요소를 필터링할 수 있습니다. 다음과 같은 XML 객체를 검토하십시오.

```
var x:XML =
  <employeeList>
    <employee id="347">
      <lastName>Zmed</lastName>
      <firstName>Sue</firstName>
      <position>Data analyst</position>
    </employee>
    <employee id="348">
      <lastName>McGee</lastName>
      <firstName>Chuck</firstName>
      <position>Jr. data analyst</position>
    </employee>
  </employeeList>
```

다음 표현식은 모두 유효합니다.

- `x.employee.(lastName == "McGee")` - 두 번째 `employee` 노드입니다.
- `x.employee.(lastName == "McGee").firstName` - 두 번째 `employee` 노드의 `firstName` 속성입니다.
- `x.employee.(lastName == "McGee").@id` - 두 번째 `employee` 노드의 `id` 특성 값입니다.
- `x.employee.(@id == 347)` - 첫 번째 `employee` 노드입니다.
- `x.employee.(@id == 347).lastName` - 첫 번째 `employee` 노드의 `lastName` 속성입니다.
- `x.employee.(@id > 300)` - 두 `employee` 속성이 모두 포함된 XMLList입니다.
- `x.employee.(position.toString().search("analyst") > -1)` - 두 `position` 속성이 모두 포함된 XMLList입니다.

존재하지 않는 특성 또는 요소를 필터링하려고 하면 Adobe Flash Player에서 예외가 발생합니다. 예를 들어, 다음 코드의 경우 두 번째 `p` 요소에 `id` 특성이 없으므로 마지막 줄에서 오류가 발생합니다.

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
         <p>Hello.</p>
       </body>;
trace(doc.p.(@id == '123'));
```

마찬가지로 다음 코드에서도 두 번째 `p` 요소의 `b` 속성이 없으므로 마지막 줄에서 오류가 발생합니다.

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
```

```
        <p>Hello.</p>
      </body>;
trace(doc.p.(b == 'Bob'));
```

이러한 오류가 발생하지 않게 하려면 다음 코드와 같이 `attribute()` 및 `elements()` 메서드를 사용하여 대응하는 특성 또는 요소가 있는 속성을 식별해야 합니다.

```
var doc:XML =
      <body>
        <p id='123'>Hello, <b>Bob</b>.</p>
        <p>Hello.</p>
      </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

다음 코드와 같이 `hasOwnProperty()` 메서드를 사용할 수도 있습니다.

```
var doc:XML =
      <body>
        <p id='123'>Hello, <b>Bob</b>.</p>
        <p>Hello.</p>
      </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

for..in 및 for each..in 명령문 사용

ActionScript 3.0에는 XMLList 객체를 반복할 수 있도록 `for..in` 및 `for each..in` 명령문이 포함되어 있습니다. 예를 들어, `myXML`이라는 XML 객체와 `myXML.item`이라는 XMLList 객체가 있다고 가정해 봅니다. 이때 XMLList 객체 `myXML.item`은 XML 객체의 두 `item` 노드로 구성되어 있습니다.

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2' quantity='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>;
```

`for..in` 명령문을 사용하면 XMLList의 속성 이름 집합을 반복할 수 있습니다.

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
  total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

for each..in 명령문을 사용하면 XMLList의 속성을 반복할 수 있습니다.

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
  total2 += prop.@quantity * prop.price;
}
```

XML 네임스페이스 사용

XML 객체 또는 문서의 네임스페이스는 해당 객체에 포함된 데이터의 유형을 식별합니다. 예를 들어, SOAP 메시징 프로토콜을 사용하는 웹 서비스에 XML 데이터를 전송하는 경우 다음과 같이 XML의 여는 태그에 네임스페이스를 선언합니다.

```
var message:XML =
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:w="http://www.test.com/weather/">
      <w:getWeatherResponse>
        <w:tempurature >78</w:tempurature>
      </w:getWeatherResponse>
    </soap:Body>
  </soap:Envelope>;
```

이 네임스페이스에는 접두어 soap와 네임스페이스를 정의하는 URI http://schemas.xmlsoap.org/soap/envelope/이 포함되어 있습니다.

ActionScript 3.0에는 XML 네임스페이스를 사용하여 작업할 수 있도록 Namespace 클래스가 포함되어 있습니다. 위의 예제에 제공된 XML 객체의 경우 다음과 같이 Namespace 클래스를 사용할 수 있습니다.

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // 출력 : http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

XML 클래스에는 네임스페이스를 사용하여 작업할 수 있도록 addNamespace(), inScopeNamespaces(), localName(), name(), namespace(), namespaceDeclarations(), removeNamespace(), setLocalName(), setName() 및 setNamespace() 등의 메서드가 포함되어 있습니다.

default xml namespace 지시문을 사용하면 XML 객체의 기본 네임스페이스를 지정할 수 있습니다. 다음 예제 코드에서는 x1과 x2 둘 다에 동일한 기본 네임스페이스가 지정되어 있습니다.

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML 유형 변환

XML 객체 및 XMLList 객체를 문자열 값으로 변환할 수 있습니다. 마찬가지로 문자열을 XML 객체 및 XMLList 객체로 변환할 수도 있습니다. 또한 모든 XML 특성 값, 이름, 텍스트 값 등은 문자열이라는 사실을 기억하십시오. 다음 단원에서는 이러한 모든 XML 유형 변환 형식에 대해 설명합니다.

XML 및 XMLList 객체를 문자열로 변환

XML 및 XMLList 클래스에는 `toString()` 메서드와 `toXMLString()` 메서드가 포함되어 있습니다. `toXMLString()` 메서드는 XML 객체의 모든 태그, 특성, 네임스페이스 선언 및 내용이 포함된 문자열을 반환합니다. 복잡한 내용, 즉 자식 요소를 포함하는 XML 객체의 경우 `toString()` 메서드는 `toXMLString()` 메서드와 완전히 동일하고, 간단한 내용, 즉 텍스트 요소 하나만 포함하는 XML 객체의 경우 `toString()` 메서드는 다음 예제와 같이 해당 요소의 텍스트 내용만 반환합니다.

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName.toXMLString());
  // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
  // burger
```

이 코드와 같이 `toString()` 또는 `toXMLString()`을 지정하지 않고 `trace()` 메서드를 사용하면 기본적으로 `toString()` 메서드를 사용하여 데이터가 변환됩니다.

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;
```

```
trace(myXML.item[0].menuName);
  // burger
```

trace() 메서드를 사용하여 코드를 디버깅하는 경우에는 trace() 메서드에서 좀 더 완전한 데이터를 출력하도록 toXMLString() 메서드를 사용할 수 있습니다.

문자열을 XML 객체로 변환

다음과 같이 new XML() 생성자를 사용하여 문자열에서 XML 객체를 만들 수 있습니다.

```
var x:XML = new XML("<a>test</a>");
```

유효하지 않거나 형식이 잘못된 XML을 나타내는 문자열을 XML로 변환하려고 하면 다음과 같이 런타임 오류가 발생합니다.

```
var x:XML = new XML("<a>test"); // 오류를 throw 합니다 .
```

문자열 형식의 특성 값, 이름 및 텍스트 값 변환

모든 XML 특성 값, 이름 및 텍스트 값은 String 데이터 유형이며 이러한 값을 다른 데이터 유형으로 변환해야 할 수 있습니다. 예를 들어, 다음 코드에서는 Number() 함수를 사용하여 텍스트 값을 숫자로 변환합니다.

```
var myXML:XML =
              <order>
                <item>
                  <price>3.95</price>
                </item>
                <item>
                  <price>1.00</price>
                </item>
              </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
  myXML.total.children()[0] = Number(myXML.total.children()[0])
                            + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

이 코드에서 Number() 함수를 사용하지 않은 경우에는 + 연산자를 문자열 연결 연산자로 해석하므로 마지막 줄의 trace() 메서드에 의해 다음과 같이 출력됩니다.

```
01.003.95
```

외부 XML 문서 읽기

URLLoader 클래스를 사용하면 URL에서 XML 데이터를 로드할 수 있습니다. 응용 프로그램에서 다음 코드를 사용하려면 예제에 나오는 XML_URL 값을 올바른 URL로 바꾸십시오.

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
  myXML = XML(myLoader.data);
  trace("Data loaded.");
}
```

XMLSocket 클래스를 사용하여 서버와의 비동기 XML 소켓 연결을 설정할 수도 있습니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

예제: 인터넷에서 RSS 데이터 로드

RSSViewer 샘플 응용 프로그램에서는 다음과 같이 ActionScript에서 XML을 사용하여 작업할 수 있는 다양한 기능을 보여 줍니다.

- XML 메서드를 사용하여 RSS 피드 형식으로 XML 데이터를 순회합니다.
- XML 메서드를 사용하여 XML 데이터를 텍스트 필드에 사용하기 위해 HTML 형식으로 어셈블합니다.

RSS 포맷은 XML을 통해 뉴스를 게시하는 데 주로 사용됩니다. 다음은 간단한 RSS 데이터 파일입니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title>Alaska - Weather</title>
  <link>http://www.nws.noaa.gov/alerts/ak.html</link>
  <description>Alaska - Watches, Warnings and Advisories</description>

  <item>
    <title>
      Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
```

```
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
  <item>
    <title>
      Short Term Forecast - Haines Borough (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
</channel>
</rss>
```

SimpleRSS 응용 프로그램은 인터넷에서 RSS 데이터를 읽은 후 헤드라인(제목), 링크 및 설명을 위해 데이터를 파싱한 다음 해당 데이터를 반환합니다. SimpleRSSUI 클래스는 UI를 제공하고, 모든 XML 처리 작업을 실행하는 SimpleRSS 클래스를 호출합니다.

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. RSSViewer 응용 프로그램 파일은 Samples/RSSViewer 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
RSSViewer.mxml 또는 RSSViewer.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/rssViewer/RSSParser.as	E4X를 사용하여 RSS(XML) 데이터를 순회하고 이에 해당하는 HTML 표현을 생성하는 메서드가 포함된 클래스입니다.
RSSData/ak.rss	샘플 RSS 파일입니다. 이 응용 프로그램은 Adobe에서 호스팅하는 Flex RSS 피드의 웹에서 RSS 데이터를 읽도록 설정되어 있습니다. 그러나 Flex RSS 피드의 스키마와 약간 다른 스키마를 사용하는 이 문서에서 RSS 데이터를 읽도록 응용 프로그램을 쉽게 변경할 수 있습니다.

XML 데이터 읽기 및 파싱

RSSParser 클래스에는 `rssXML` 변수에 저장된 입력 RSS 데이터를 HTML 포맷 출력(`rssOutput`)이 들어 있는 문자열로 변환하는 `xmlLoaded()` 메서드가 포함되어 있습니다.

소스 RSS 데이터에 기본 네임스페이스가 포함되어 있는 경우 코드에서는 메서드의 시작 부분에서 기본 XML 네임스페이스를 설정합니다.

```
if (rssXML.namespace("") != undefined)
{
  default xml namespace = rssXML.namespace("");
}
```

그러면 다음 줄에서 소스 XML 데이터의 내용을 반복하여 `item`이라는 각 하위 속성을 검사합니다.

```
for each (var item:XML in rssXML..item)
{
  var itemTitle:String = item.title.toString();
  var itemDescription:String = item.description.toString();
  var itemLink:String = item.link.toString();
  outXML += buildItemHTML(itemTitle,
                itemDescription,
                itemLink);
}
```

처음 세 줄에서는 문자열 변수를 설정하여 XML 데이터에 대한 `item` 속성의 제목, 설명 및 링크 속성을 나타냅니다. 그러면 다음 줄에서 `buildItemHTML()` 메서드를 호출하여 HTML 데이터를 XMLList 객체 형식으로 가져옵니다. 이때 세 개의 새 문자열 변수를 매개 변수로 사용합니다.

XMLList 데이터 어셈블

HTML 데이터(XMLList 객체)는 다음과 같은 형식으로 되어 있습니다.

```
<b>itemTitle</b>
<p>
  itemDescription
  <br />
  <a href="link">
    <font color="#008000">More...</font>
  </a>
</p>
```

메서드의 첫 번째 줄에서는 기본 xml 네임스페이스를 제거합니다.

```
default xml namespace = new Namespace();
```

`default xml namespace` 지시문에는 함수 블록 레벨 범위가 있습니다. 즉, 이 선언의 범위가 `buildItemHTML()` 메서드입니다.

그 다음에 나오는 줄에서는 함수에 전달된 문자열 인수에 따라 XMLList를 어셈블합니다.

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
     // <font color="#008000"></font></a>
     // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

이 XMLList 객체는 ActionScript HTML 텍스트 필드에 적합한 문자열 데이터를 나타냅니다.

`xmlLoaded()` 메서드는 `buildItemHTML()` 메서드의 반환값을 사용하고 이 값을 문자열로 변환합니다.

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

RSS 피드 제목 추출 및 사용자 정의 이벤트 보내기

`xmlLoaded()` 메서드는 소스 RSS XML 데이터의 정보를 기초로 `rssTitle` 문자열 변수를 설정합니다.

```
rssTitle = rssXML.channel.title.toString();
```

마지막으로 `xmlLoaded()` 메서드는 데이터가 파싱되어 사용할 수 있음을 응용 프로그램에 알리는 이벤트를 생성합니다.

```
       dataWritten = new Event("dataWritten", true);
```

제 12 장

12 디스플레이 프로그래밍

ActionScript 3.0의 디스플레이 프로그래밍을 사용하면 Adobe Flash Player 9의 Stage에 표시되는 요소로 작업할 수 있습니다. 이 장에서는 화면 요소 작업과 관련된 기본 개념을 설명합니다. 또한 시각적 요소를 프로그래밍 방식으로 구성하는 자세한 방법과 표시 객체에 대한 사용자 정의 클래스를 만드는 방법에 대해 살펴봅니다.

목차

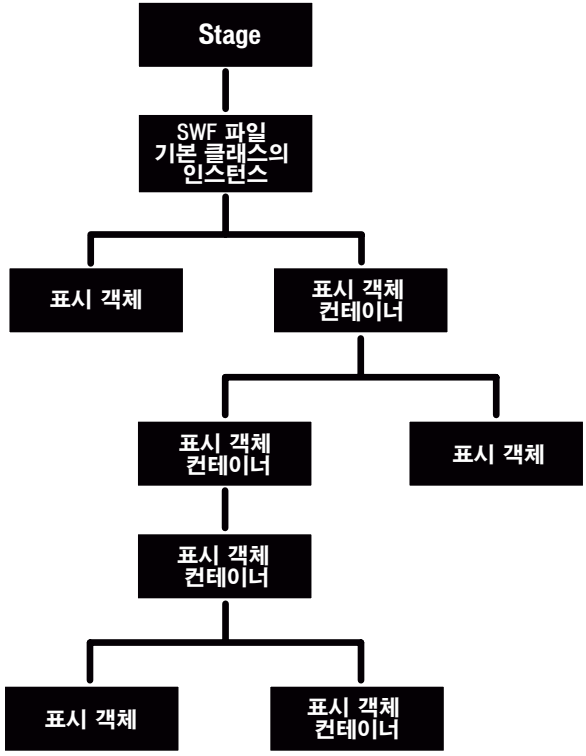
디스플레이 프로그래밍의 기초

디스플레이 프로그래밍 소개

ActionScript 3.0으로 작성된 각 응용 프로그램에는 *표시 목록*이라는 표시된 객체의 계층이 있습니다. 이 표시 목록에는 응용 프로그램에 보이는 요소가 모두 포함되어 있습니다. 표시 요소는 다음 그룹 중 하나 이상에 속합니다.

- Stage

  Stage는 표시 객체의 기본 컨테이너입니다. 각 응용 프로그램에는 모든 화면 표시 객체를 포함하는 Stage 객체가 하나 있습니다. Stage는 최상위 컨테이너이며 표시 목록 계층의 맨 위에 있습니다.



각 SWF 파일에는 *SWF 파일의 기본 클래스*라는 관련된 ActionScript 클래스가 있습니다. Flash Player는 HTML 페이지에서 SWF 파일을 열 때 해당 클래스의 생성자 함수를 호출합니다. 그러면 생성되는 인스턴스(항상 표시 객체의 유형)가 Stage 객체의 자식으로 추가됩니다. SWF 파일의 기본 클래스는 항상 Sprite 클래스를 확장합니다. 자세한 내용은 356페이지의 "표시 목록 방식의 장점"을 참조하십시오.

DisplayObject 인스턴스의 `stage` 속성을 통해 Stage에 액세스할 수 있습니다. 자세한 내용은 366페이지의 "Stage 속성 설정"을 참조하십시오.

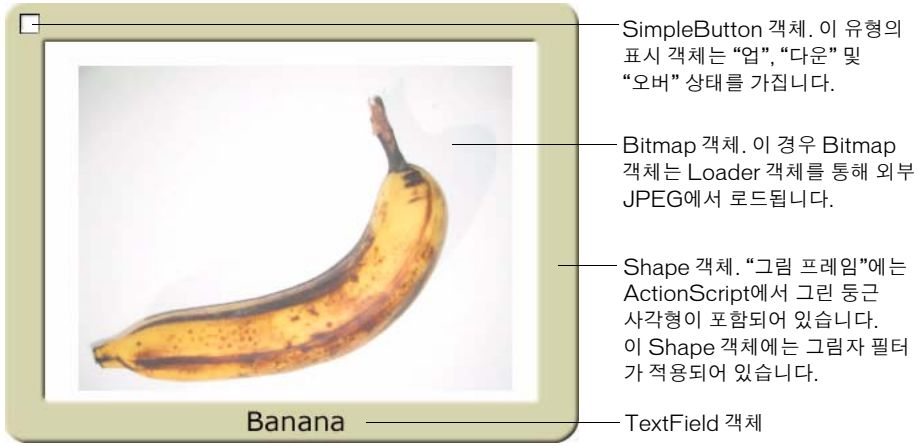
- 표시 객체

  ActionScript 3.0에서 응용 프로그램의 화면에 표시되는 모든 요소는 *표시 객체* 유형입니다. flash.display 패키지에는 많은 다른 클래스에 의해 확장되는 기본 클래스인 DisplayObject 클래스가 포함되어 있습니다. 이러한 각 클래스는 벡터 모양, 무비 클립, 텍스트 필드 등과 같은 서로 다른 유형의 표시 객체를 나타냅니다. 이러한 클래스에 대한 개요는 356페이지의 "표시 목록 방식의 장점"을 참조하십시오.

- 표시 객체 컨테이너

  표시 객체 컨테이너는 고유한 시각적 표현이 있고 자식 객체를 포함할 수 있는 특수 유형의 표시 객체입니다. 이때 자식 객체도 표시 객체입니다.

  DisplayObjectContainer 클래스는 DisplayObject 클래스의 하위 클래스입니다. DisplayObjectContainer 객체는 *자식 목록*에 여러 표시 객체를 포함할 수 있습니다. 예를 들어, 다음 그림에서는 다양한 표시 객체를 포함하는 Sprite라는 DisplayObjectContainer 객체 유형을 보여 줍니다.



표시 객체에 대해 언급할 때는 DisplayObjectContainer 객체도 *표시 객체 컨테이너* 또는 간단히 *컨테이너*라고 합니다.

모든 보이는 표시 객체는 DisplayObject 클래스에서 상속되지만 각 유형은 DisplayObject 클래스의 특정 하위 클래스입니다. 예를 들어, Shape 클래스 또는 Video 클래스에 대한 생성자 함수가 있지만, DisplayObject 클래스에 대한 생성자 함수는 없습니다.

앞에서 설명한 것처럼 Stage는 표시 객체 컨테이너입니다.

일반 디스플레이 프로그래밍 작업

대부분의 ActionScript 프로그래밍에는 시각적 요소 만들기 및 조작이 포함되므로 디스플레이 프로그래밍과 관련된 다양한 작업이 이루어집니다. 이 장에서는 다음과 같이 모든 표시 객체에 적용되는 일반 작업에 대해 설명합니다.

- 표시 목록 및 표시 객체 컨테이너 작업
  - 표시 목록에 표시 객체 추가
  - 표시 목록에서 객체 제거
  - 표시 컨테이너 간에 객체 이동
  - 다른 객체의 앞 또는 뒤로 객체 이동
- Stage 작업
  - 프레임 속도 설정
  - Stage 배율 제어
  - 전체 화면 모드 작업
- 표시 객체 이벤트 처리
- 표시 객체 위치 지정(드래그 앤 드롭 상호 작용 포함)
- 표시 객체 크기 조절, 배율 및 회전
- 표시 객체에 블렌드 모드, 색상 변환 및 투명도 적용
- 표시 객체 마스크 적용
- 표시 객체 애니메이션
- 외부 표시 내용 로드(예: SWF 파일 또는 이미지)

이 설명서의 이후 장에서는 표시 객체 사용을 위한 추가 작업에 대해 설명합니다. 이러한 작업에는 모든 표시 객체에 적용되는 작업과 특정 표시 객체 유형에 관련된 작업이 모두 포함됩니다.

- 표시 객체에서 ActionScript를 사용하여 벡터 그래픽 그리기(421페이지의 제14장, “드로잉 API 사용” 참조)
- 표시 객체에 기하학적 변형 적용(405페이지의 제13장, “기하 도형을 사용한 작업” 참조)
- 표시 객체에 그래픽 필터 효과(예: 흐림, 광선, 그림자 등) 적용(437페이지의 제15장, “표시 객체 필터링” 참조)
- MovieClip 고유 특성 작업(461페이지의 제16장, “무비 클립을 사용한 작업” 참조)

- TextField 객체 작업(477페이지의 제17장, “텍스트를 사용한 작업” 참조)
- 비트맵 그래픽 작업(505페이지의 제18장, “비트맵을 사용한 작업” 참조)
- 비디오 요소 작업(519페이지의 제19장, “비디오를 사용한 작업” 참조)

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 알파: 색상의 투명도(보다 정확하게는 불투명도)를 나타내는 색상 값입니다. 예를 들어, 알파 채널 값이 60%인 색상은 전체 강도의 60%만 표시되고 40%는 투명하게 나타납니다.
- 비트맵 그래픽: 컴퓨터에서 색상 픽셀의 격자(행, 열)로 정의되는 그래픽입니다. 일반적으로 비트맵 그래픽에는 디지털 사진 및 이와 유사한 이미지가 포함됩니다.
- 블렌딩 모드: 겹치는 두 개의 이미지 내용이 서로 영향을 미치는 방식을 지정하는 것입니다. 일반적으로 불투명한 이미지가 다른 이미지 위에 놓이게 되면 아래쪽의 이미지를 가려서 아래쪽 이미지가 전혀 보이지 않지만, 블렌딩 모드를 다르게 하면 이미지 색상의 블렌드 방식이 변경되어 두 개 이미지가 혼합되어 나타납니다.
- 표시 목록: Flash Player에 의해 가시 화면 내용으로 렌더링되는 표시 객체 계층 구조입니다. Stage는 표시 목록의 루트이며, Stage 또는 해당 자식 중 하나에 연결된 모든 표시 객체가 표시 목록을 구성합니다(객체가 Stage 경계 외부에 있는 등 실제로 렌더링되지 않은 경우도 포함).
- 표시 객체: Flash Player의 시각적 내용 중 일부 유형을 나타내는 객체입니다. 표시 객체만 표시 목록에 포함될 수 있으며 모든 표시 객체 클래스는 DisplayObject 클래스의 하위 클래스입니다.
- 표시 객체 컨테이너: 일반적으로 시각적 표현을 가지며 자식 표시 객체를 포함할 수 있는 특수한 유형의 표시 객체입니다.
- SWF 파일의 기본 클래스: SWF 파일에서 가장 바깥쪽 표시 객체의 비헤이비어를 정의하는 클래스로, 개념적으로 SWF 파일 자체의 클래스를 나타냅니다. 예를 들어, Flash 제작에서 만들어진 SWF에는 다른 모든 타임라인이 포함된 “기본 타임라인”이 있으며, SWF 파일의 기본 클래스는 기본 타임라인이 인스턴스인 클래스가 됩니다.
- 마스크 적용: 이미지의 특정 부분이 보이지 않도록 숨기는, 즉 이미지의 특정 부분만 표시되도록 하는 기술입니다. 이미지의 숨겨진 부분은 투명하게 되므로 아래쪽 내용도 볼 수 있습니다. 이 용어는 특정 영역을 칠하지 않을 때 사용하는 마스킹 테이프와 관계 있습니다.
- Stage: SWF의 모든 시각적 내용의 기반이나 배경이 되는 시각적 컨테이너입니다.
- 변형: 객체 회전, 크기 변경, 모양 기울이기 또는 왜곡, 색상 변경 등 그래픽의 시각적 특징을 조절합니다.
- 벡터 그래픽: 컴퓨터에서 특정 특성(두께, 길이, 크기, 각도, 위치)으로 그려진 선 및 모양으로 정의되는 그래픽입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장에서는 시각적인 내용의 작성 및 조작에 대해 다루기 때문에 기본적으로 이 장의 모든 코드 샘플은 시각적 객체를 만들어 화면에 표시하며, 샘플을 테스트할 경우 이전 장의 변수 값 대신 Flash Player에서 결과를 확인해야 합니다. 이 장의 코드 샘플을 테스트하려면:

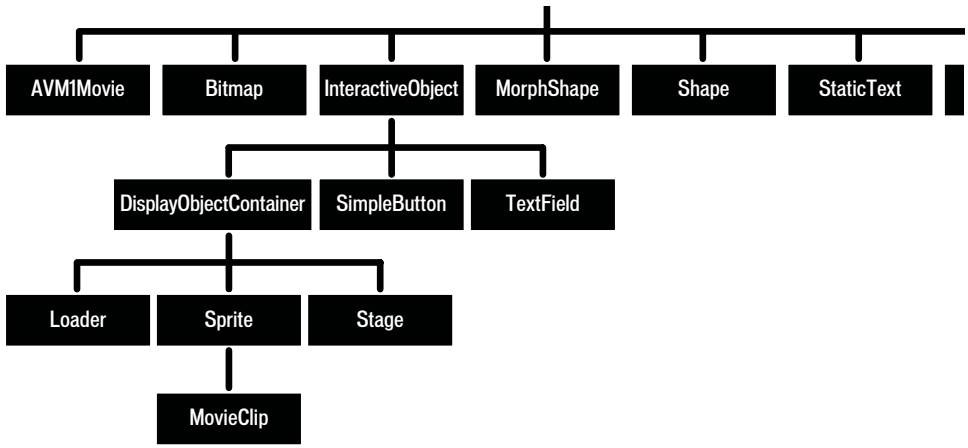
1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   화면에 표시된 코드 결과를 확인합니다. 모든 `trace()` 함수 호출이 [출력] 패널에 표시됩니다.

예제 코드 샘플 테스트와 관련된 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

기본 표시 클래스

ActionScript 3.0 flash.display 패키지에는 Flash Player에서 표시될 수 있는 시각적 객체에 대한 클래스가 포함되어 있습니다. 다음 그림에서는 이러한 기본 표시 객체 클래스의 하위 클래스 관계를 보여 줍니다.



이 그림에서는 표시 객체 클래스의 클래스 상속을 보여 줍니다. StaticText, TextField 및 Video와 같은 일부 클래스는 flash.display 패키지에 없지만, DisplayObject 클래스에서 여전히 상속됩니다.

DisplayObject 클래스를 확장하는 모든 클래스는 해당 메서드와 속성을 상속합니다. 자세한 내용은 359페이지의 "DisplayObject 클래스의 속성 및 메서드"를 참조하십시오.

flash.display 패키지에 포함된 다음 클래스의 객체를 인스턴스화할 수 있습니다.

- Bitmap - Bitmap 클래스를 사용하여 외부 파일에서 로드되거나 ActionScript를 통해 렌더링되는 비트맵 객체를 정의할 수 있습니다. Loader 클래스를 통해 외부 파일에서 비트맵을 로드할 수 있습니다. GIF, JPG 또는 PNG 파일을 로드할 수 있습니다. 또한 사용자 정의 데이터를 사용하여 BitmapData 객체를 만든 다음 해당 데이터를 사용하는 Bitmap 객체를 만들 수 있습니다. BitmapData 클래스의 메서드를 사용하면 ActionScript에서 로드하거나 만든 비트맵을 변경할 수 있습니다. 자세한 내용은 394페이지의 "표시 객체 로드" 및 505페이지의 제18장, "비트맵을 사용한 작업"을 참조하십시오.
- Loader - Loader 클래스를 사용하여 외부 에셋(SWF 파일 또는 그래픽)을 로드할 수 있습니다. 자세한 내용은 394페이지의 "표시 내용을 동적으로 로드"를 참조하십시오.
- Shape - Shape 클래스를 사용하여 사각형, 선, 원 등과 같은 벡터 그래픽을 만들 수 있습니다. 자세한 내용은 421페이지의 제14장, "드로잉 API 사용"을 참조하십시오.
- SimpleButton - SimpleButton 객체는 Flash 버튼 심볼을 ActionScript로 표현한 것입니다. SimpleButton 인스턴스에는 업, 다운, 오버의 세 가지 버튼 상태가 있습니다.
- Sprite - Sprite 객체는 자체 그래픽과 자식 표시 객체를 포함할 수 있습니다. Sprite 클래스는 DisplayObjectContainer 클래스를 확장합니다. 자세한 내용은 360페이지의 "표시 객체 컨테이너 작업" 및 421페이지의 제14장, "드로잉 API 사용"을 참조하십시오.
- MovieClip - MovieClip 객체는 Flash 제작 도구로 만든 무비 클립 심볼의 ActionScript 형식입니다. 실제로 MovieClip은 타임라인이 있다는 점을 제외하고 Sprite 객체와 비슷합니다. 자세한 내용은 461페이지의 제16장, "무비 클립을 사용한 작업"을 참조하십시오.

flash.display 패키지에 없는 다음 클래스는 DisplayObject 클래스의 하위 클래스입니다.

- flash.text 패키지에 포함된 TextField 클래스는 텍스트 표시 및 입력을 위한 표시 객체입니다. 자세한 내용은 477페이지의 제17장, "텍스트를 사용한 작업"을 참조하십시오.
- flash.media 패키지에 포함된 Video 클래스는 비디오 파일을 표시하는 데 사용되는 표시 객체입니다. 자세한 내용은 519페이지의 제19장, "비디오를 사용한 작업"을 참조하십시오.

flash.display 패키지에 있는 다음 클래스는 DisplayObject 클래스를 확장하지만 해당 클래스의 인스턴스를 만들 수 없습니다. 이러한 클래스는 공통 기능을 하나의 클래스로 결합하여 다른 표시 객체에 대한 부모 클래스 역할을 합니다.

- AVM1Movie - AVM1Movie 클래스는 ActionScript 1.0 및 2.0에서 만든 로드된 SWF 파일을 나타내는 데 사용됩니다.
- DisplayObjectContainer - Loader, Stage, Sprite 및 MovieClip 클래스는 각각 DisplayObjectContainer 클래스를 확장합니다. 자세한 내용은 360페이지의 "표시 객체 컨테이너 작업"을 참조하십시오.

- InteractiveObject - InteractiveObject 클래스는 마우스 및 키보드와 상호 작용하는 데 사용되는 모든 객체에 대한 기본 클래스입니다. SimpleButton, TextField, Video, Loader, Sprite, Stage 및 MovieClip 객체는 모두 InteractiveObject 클래스의 하위 클래스입니다. 마우스 및 키보드 상호 작용 만들기에 대한 자세한 내용은 589페이지의 제21장, “사용자 입력 캡처”를 참조하십시오.
- MorphShape - 이러한 객체는 Flash 제작 도구에서 모양 트윈을 만들 때 만들어집니다. 이러한 객체는 ActionScript를 사용하여 인스턴스화할 수 없지만 표시 목록에서 액세스할 수 있습니다.
- Stage - Stage 클래스는 DisplayObjectContainer 클래스를 확장합니다. 하나의 응용 프로그램에는 표시 목록 계층의 맨 위에 하나의 Stage 인스턴스가 있습니다. Stage에 액세스하려면 DisplayObject 인스턴스의 `stage` 속성을 사용합니다. 자세한 내용은 366페이지의 “Stage 속성 설정”을 참조하십시오.

또한 flash.text 패키지의 StaticText 클래스는 DisplayObject 클래스를 확장하지만, 코드에 그 인스턴스를 만들 수 없습니다. 정적 텍스트 필드는 Adobe Flash CS3 Professional에서만 만들 수 있습니다.

표시 목록 방식의 장점

ActionScript 3.0에는 표시 객체 유형마다 별도의 클래스가 있습니다. ActionScript 1.0 및 2.0에는 많은 동일한 유형의 객체가 MovieClip 클래스 하나에 모두 포함되어 있습니다.

이 클래스 개별화 및 표시 목록 계층 구조에는 다음과 같은 장점이 있습니다.

- 보다 효율적인 렌더링 및 메모리 사용 감소
- 심도 관리 향상
- 표시 목록 전체 탐색
- 목록 외 표시 객체
- 보다 쉬운 표시 객체 하위 클래스화

이러한 장점에 대해서는 다음 단원에서 설명합니다.

보다 효율적인 렌더링 및 파일 크기 축소

ActionScript 1.0 및 2.0에서는 MovieClip 객체에서만 모양을 그릴 수 있습니다. ActionScript 3.0에는 모양을 그릴 수 있는 간단한 표시 객체 클래스가 있습니다. 이러한 ActionScript 3.0 표시 객체 클래스는 MovieClip 객체에 포함되는 전체 메서드와 속성을 포함하지 않기 때문에, 메모리 및 프로세서 리소스 사용을 줄일 수 있습니다.

예를 들어, 각 MovieClip 객체는 무비 클립의 타임라인 속성을 포함하지만 Shape 객체는 이 속성을 포함하지 않습니다. 타임라인 관리를 위한 속성은 많은 메모리와 프로세서 리소스를 사용할 수 있습니다. ActionScript 3.0에서는 Shape 객체를 사용하여 성능을 높일 수 있습니다. Shape 객체는 복잡한 MovieClip 객체보다 오버헤드가 적습니다. Flash Player에서 사용되지 않는 MovieClip 속성을 관리할 필요가 없으므로, 속도가 향상되고 객체에서 사용하는 메모리 용량도 감소됩니다.

심도 관리 개선

ActionScript 1.0 및 2.0에서는 심도가 `getNextHighestDepth()`와 같은 선형 심도 관리 스키마와 메서드를 통해 관리됩니다.

ActionScript 3.0에는 표시 객체의 심도를 관리하는 보다 편리한 메서드와 속성을 가진 DisplayObjectContainer 클래스가 포함되어 있습니다.

ActionScript 3.0에서는 표시 객체를 DisplayObjectContainer 인스턴스의 자식 목록에서 새 위치로 이동하면 표시 객체 컨테이너에 있는 다른 자식 객체들의 위치가 자동으로 다시 배치되고 표시 객체 컨테이너의 적절한 자식 인덱스 위치에 할당됩니다.

또한 ActionScript 3.0에서는 표시 객체 컨테이너의 모든 자식 객체를 항상 검색할 수 있습니다. 모든 DisplayObjectContainer 인스턴스에는 표시 객체 컨테이너에 있는 자식 수를 나열하는 `numChildren` 속성이 있습니다. 표시 객체 컨테이너의 자식 목록에는 항상 인덱스가 지정되어 있으므로, 인덱스 위치 0부터 마지막 인덱스 위치(`numChildren - 1`)까지 목록의 모든 객체를 검사할 수 있습니다. 이 작업은 ActionScript 1.0 및 2.0에 있는 MovieClip 객체의 메서드와 속성으로는 가능하지 않습니다.

ActionScript 3.0에서는 표시 객체 컨테이너 자식 목록의 인덱스 번호에 빠진 번호가 없기 때문에 표시 목록을 차례대로 쉽게 탐색할 수 있습니다. ActionScript 1.0 및 2.0에서보다 표시 목록을 탐색하고 객체 심도를 관리하기가 훨씬 쉬워졌습니다. ActionScript 1.0 및 2.0에서는 무비 클립이 심도 순서에 간격을 두고 객체를 포함할 수 있기 때문에 객체 목록을 탐색하기가 어려울 수 있습니다. ActionScript 3.0에서는 표시 객체 컨테이너의 각 자식 목록이 내부적으로 단일 배열로 캐시되므로 인덱스를 기준으로 매우 빠르게 조회할 수 있습니다. 표시 객체 컨테이너의 모든 자식을 매우 빠르게 반복할 수도 있습니다.

ActionScript 3.0에서는 DisplayObjectContainer 클래스의 `getChildByName()` 메서드를 사용하여 표시 객체 컨테이너의 자식에 액세스할 수도 있습니다.

표시 목록 전체 탐색

ActionScript 1.0 및 2.0에서는 Flash 제작 도구에서 그린 벡터 모양과 같은 일부 객체에 액세스할 수 없습니다. ActionScript 3.0에서는 ActionScript에서 만든 객체와 Flash 제작 도구에서 만든 모든 표시 객체를 포함하여 표시 목록에 있는 모든 객체에 액세스할 수 있습니다. 자세한 내용은 364페이지의 "표시 목록 탐색"을 참조하십시오.

목록 외 표시 객체

ActionScript 3.0에서는 보이는 표시 목록에 없는 표시 객체를 만들 수 있습니다. 이러한 표시 객체를 *목록 외* 표시 객체라고 합니다. 표시 객체는 표시 목록에 이미 추가된 DisplayObjectContainer 인스턴스의 `addChild()` 또는 `addChildAt()` 메서드를 호출할 경우에만 보이는 표시 목록에 추가됩니다.

목록 외 표시 객체를 사용하면, 여러 표시 객체가 포함된 표시 객체 컨테이너가 여러 개 있는 객체처럼 복잡한 표시 객체를 모을 수 있습니다. 표시 객체를 목록 외 상태로 유지하면, 이러한 표시 객체를 렌더링하는 데 처리 시간을 사용하지 않고도 복잡한 객체를 모을 수 있습니다. 그런 다음 필요에 따라 표시 목록에 목록 외 표시 객체를 추가할 수 있습니다. 또한 표시 객체 컨테이너의 자식을 표시 목록 내외 및 표시 목록 내의 원하는 위치로 이동할 수 있습니다.

보다 쉬운 표시 객체 하위 클래스화

ActionScript 1.0 및 2.0에서는 기본 모양을 만들거나 비트맵을 표시하려면 SWF 파일에 새 MovieClip 객체를 추가해야 합니다. ActionScript 3.0에는 DisplayObject 클래스에 Shape, Bitmap 등과 같은 많은 내장 하위 클래스가 포함되어 있습니다. ActionScript 3.0의 클래스는 특정 객체 유형에 대해 더 구체화되어, 내장 클래스의 기본 하위 클래스를 쉽게 만들 수 있습니다.

예를 들어, ActionScript 2.0에서 원을 그리려면 사용자 정의 클래스 객체가 인스턴스화될 때 MovieClip 클래스를 확장하는 CustomCircle 클래스를 만들 수 있습니다. 그러나, 이 클래스는 해당 클래스에 적용되지 않는 MovieClip 클래스의 많은 속성과 메서드(예: `totalFrames`)를 포함하고 있습니다. ActionScript 3.0에서는 Shape 객체를 확장하는 CustomCircle 클래스를 만들 수 있으므로, MovieClip 클래스에 포함되어 있지만 관련 없는 속성과 메서드는 이 클래스에 포함되지 않습니다. 다음 코드는 CustomCircle 클래스의 예를 보여 줍니다.

```
import flash.display.*;

private class CustomCircle extends Shape
{
```

```
  var xPos:Number;
  var yPos:Number;
  var radius:Number;
  var color:uint;
  public function CustomCircle(xInput:Number,
                      yInput:Number,
                      rInput:Number,
                      colorInput:uint)
  {
    xPos = xInput;
    yPos = yInput;
    radius = rInput;
    color = colorInput;
    this.graphics.beginFill(color);
    this.graphics.drawCircle(xPos, yPos, radius);
  }
}
```

표시 객체 작업

지금까지 Stage, 표시 객체, 표시 객체 컨테이너 및 표시 목록의 기본 개념에 대해 살펴보았으며, 이 단원에서는 ActionScript 3.0을 사용한 표시 객체 작업에 대해 보다 자세한 정보를 제공합니다.

DisplayObject 클래스의 속성 및 메서드

모든 표시 객체는 DisplayObject 클래스의 하위 클래스이므로 DisplayObject 클래스의 속성과 메서드를 상속합니다. 상속되는 속성은 모든 표시 객체에 적용되는 기본 속성입니다. 예를 들어, 각 표시 객체에는 표시 객체 컨테이너에서 객체의 위치를 지정하는 `x` 속성과 `y` 속성이 있습니다.

DisplayObject 클래스 생성자를 사용하여 DisplayObject 인스턴스를 만들 수 없습니다. `new` 연산자를 사용하여 객체를 인스턴스화하려면 Sprite와 같은 다른 유형의 객체(DisplayObject 클래스의 하위 클래스인 객체)를 만들어야 합니다. 또한 사용자 정의 표시 객체 클래스를 만들려면 Shape 클래스 또는 Sprite 클래스처럼 사용 가능한 생성자 함수가 있는 표시 객체 하위 클래스 중 하나의 하위 클래스를 만들어야 합니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 DisplayObject 클래스 설명을 참조하십시오.

표시 목록에 표시 객체 추가

표시 객체를 인스턴스화할 경우 표시 목록에 있는 표시 객체 컨테이너에 표시 객체 인스턴스를 추가할 때까지는 해당 표시 객체가 Stage의 화면에 표시되지 않습니다. 예를 들어, 다음 코드에서 `myText` TextField 객체는 코드의 마지막 행을 생략하면 표시되지 않습니다. 코드의 마지막 행에서 `this` 키워드는 표시 목록에 이미 추가된 표시 객체 컨테이너를 나타내야 합니다.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

스테이지에 시각적 요소를 추가하면 해당 요소가 Stage 객체의 *자식*이 됩니다. 응용 프로그램에 로드되는 첫 번째 SWF 파일(예: HTML 페이지에 포함된 파일)은 Stage 객체의 자식으로 자동 추가됩니다. Sprite 클래스를 확장하는 모든 유형의 객체가 자식이 될 수 있습니다.

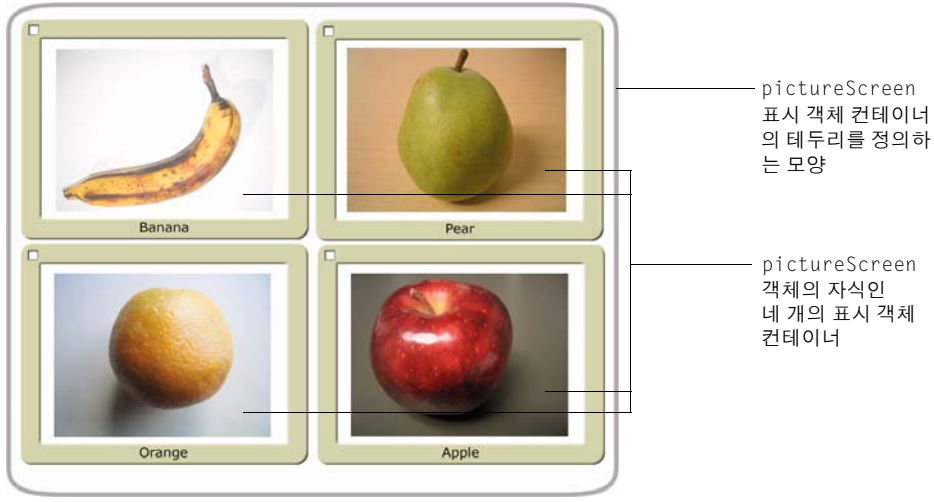
ActionScript를 사용하지 *않고*(예: Adobe Flex Builder 2에서 MXML 태그를 추가하거나 Flash의 Stage에 항목을 배치하는 방법) 만든 모든 표시 객체가 표시 목록에 추가됩니다. 이러한 표시 객체를 ActionScript를 통해 추가하지 않지만 ActionScript를 통해 액세스할 수 있습니다. 예를 들어, 다음 코드는 ActionScript가 아니라 제작 도구에서 추가한 `button1`이라는 객체의 폭을 조절합니다.

```
button1.width = 200;
```

표시 객체 컨테이너 작업

DisplayObjectContainer 객체가 표시 목록에서 삭제되거나 다른 방식으로 이동 또는 변형될 경우 DisplayObjectContainer의 각 표시 객체도 함께 삭제, 이동 또는 변형됩니다.

표시 객체 컨테이너는 그 자체가 표시 객체의 한 유형이므로 다른 표시 객체 컨테이너에 추가될 수 있습니다. 예를 들어, 다음 이미지는 하나의 외곽선 모양과 다른 네 개의 표시 객체 컨테이너(PictureFrame 유형)를 포함하는 표시 객체 컨테이너 pictureScreen을 보여 줍니다.



표시 객체가 표시 목록에 나타나게 하려면 표시 객체를 표시 목록에 있는 표시 객체 컨테이너에 추가해야 합니다. 이때 컨테이너 객체의 addChild() 메서드 또는 addChildAt() 메서드를 사용합니다. 예를 들어, 다음 코드의 마지막 행이 없으면 myTextField 객체가 표시되지 않습니다.

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

이 코드 목록에서 this.root는 코드를 포함하는 MovieClip 표시 객체 컨테이너를 가리킵니다. 실제 코드에서는 다른 컨테이너를 지정할 수 있습니다.

addChildAt() 메서드를 사용하여 표시 객체 컨테이너 자식 목록의 특정 위치에 자식을 추가합니다. 자식 목록에서 0부터 시작하는 인덱스 위치는 표시 객체의 레이어 위치(전후 순서)와 관련이 있습니다. 예를 들어, 다음 세 표시 객체를 생각해 볼 수 있습니다. 각 객체는 Ball이라는 사용자 정의 객체로부터 만들어졌습니다.

addChildAt() 메서드를 사용하여 컨테이너에서 이러한 표시 객체의 레이어 위치를 조정할 수 있습니다. 예를 들어, 다음과 같은 코드를 살펴봅니다.

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

이 코드를 실행하면 container DisplayObjectContainer 객체에서 표시 객체가 다음과 같이 배치됩니다. 객체의 레이어 위치에 주목하십시오.

객체의 위치를 표시 목록의 맨 위로 이동하려면 목록에 해당 객체를 다시 추가하면 됩니다. 예를 들어, 이전 코드를 실행한 후에 ball_A를 스택의 맨 위로 이동하려면 다음 코드 행을 사용합니다.

```
container.addChild(ball_A);
```

이 코드는 container 표시 목록의 해당 위치에서 ball_A를 효과적으로 제거한 후 목록의 맨 위에 다시 추가합니다. 그러면 해당 객체가 스택의 맨 위로 이동합니다.

getChildAt() 메서드를 사용하여 표시 객체의 레이어 순서를 확인할 수 있습니다. getChildAt() 메서드는 전달된 인덱스 번호를 기반으로 컨테이너의 자식 객체를 반환합니다. 예를 들어, 다음 코드는 container DisplayObjectContainer 객체의 자식 목록에서 서로 다른 위치에 있는 표시 객체의 이름을 표시합니다.

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

부모 컨테이너의 자식 목록에서 표시 객체를 제거하면 목록에서 더 높은 위치에 있는 요소가 자식 인덱스에서 한 위치씩 아래로 이동합니다. 예를 들어, 이전 코드에서 다음 코드는 자식 목록에서 하위에 있는 표시 객체를 제거함에 따라 `container` DisplayObjectContainer의 위치 2에 있는 표시 객체가 위치 1로 어떻게 이동하는지 보여 줍니다.

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

`removeChild()` 및 `removeChildAt()` 메서드는 표시 객체 인스턴스를 완전히 삭제하지 않고, 컨테이너의 자식 목록에서만 제거합니다. 따라서 다른 변수가 인스턴스를 계속 참조할 수 있습니다. 객체를 완전히 제거하려면 `delete` 연산자를 사용합니다.

표시 객체는 부모 컨테이너가 하나뿐이므로, 표시 객체 인스턴스를 하나의 표시 객체 컨테이너에만 추가할 수 있습니다. 예를 들어, 다음 코드는 표시 객체 `tf1`이 하나의 컨테이너(이 경우 DisplayObjectContainer 클래스를 확장하는 Sprite)에만 존재할 수 있음을 보여 줍니다.

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // 텍스트 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // 텍스트 1
```

한 표시 객체 컨테이너에 포함된 표시 객체를 다른 표시 객체 컨테이너에 추가하면 해당 표시 객체가 첫 번째 표시 객체 컨테이너의 자식 목록에서 제거됩니다.

위에서 설명한 메서드 외에도 DisplayObjectContainer 클래스는 자식 표시 객체 작업을 위한 다음과 같은 다양한 메서드를 정의합니다.

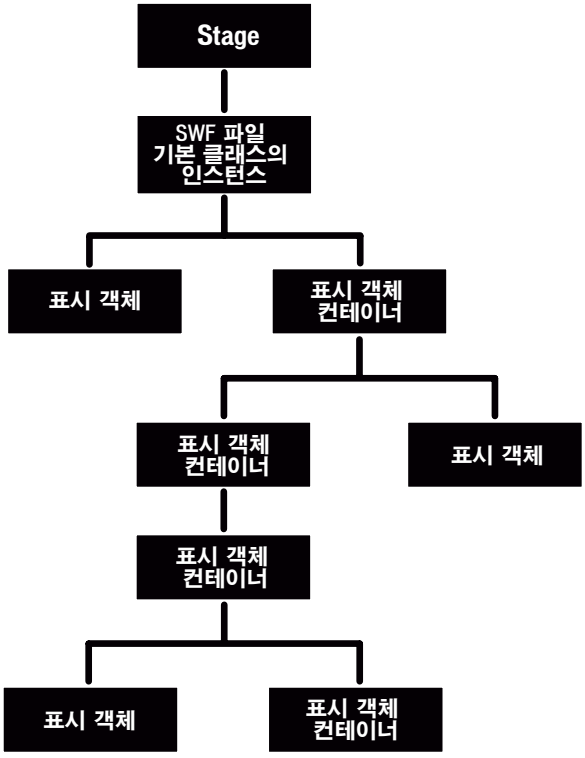
- `contains()`: 표시 객체가 DisplayObjectContainer의 자식인지 여부를 결정합니다.
- `getChildByName()`: 이름을 기준으로 표시 객체를 검색합니다.
- `getChildIndex()`: 표시 객체의 인덱스 위치를 반환합니다.
- `setChildIndex()`: 자식 표시 객체의 위치를 변경합니다.
- `swapChildren()`: 두 표시 객체의 순서(전후 순서)를 교체합니다.
- `swapChildrenAt()`: 인덱스 값으로 지정된 두 표시 객체의 순서(전후 순서)를 교체합니다.

자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 관련 항목을 참조하십시오.

표시 목록에 없는 표시 객체(Stage 객체의 자식인 표시 객체 컨테이너에 포함되지 않은 표시 객체)를 *목록 외* 표시 객체라고 합니다.

표시 목록 탐색

이미 알고 있듯이 표시 목록은 트리 구조입니다. 트리의 맨 위에는 Stage가 있고 여기에는 여러 표시 객체를 포함할 수 있습니다. 자체가 표시 객체 컨테이너인 이러한 표시 객체는 다른 표시 객체나 표시 객체 컨테이너를 포함할 수 있습니다.



DisplayObjectContainer 클래스에는 표시 객체 컨테이너의 자식 목록을 사용하여 표시 목록을 탐색하기 위한 속성과 메서드가 포함되어 있습니다. 예를 들어, 두 표시 객체 `title`과 `pict`를 `container` 객체(Sprite 객체, Sprite 클래스는 DisplayObjectContainer 클래스를 확장)에 추가하는 다음 코드를 살펴봅니다.

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

`getChildAt()` 메서드는 특정 인덱스 위치에 있는 표시 목록의 자식을 반환합니다.

```
trace(container.getChildAt(0) is TextField); // true
```

이름을 기준으로 자식 객체에 액세스할 수도 있습니다. 각 표시 객체에는 **name** 속성이 있습니다. 이 속성을 할당하지 않을 경우 **Flash Player**는 `"instance1"`과 같은 기본값을 할당합니다. 예를 들어, 다음 코드는 `getChildByName()` 메서드를 사용하여 이름이 `"banana loader"`인 자식 표시 객체에 액세스하는 방법을 보여 줍니다.

```
trace(container.getChildByName("banana loader") is Loader); // true
```

`getChildByName()` 메서드를 사용하면 `getChildAt()` 메서드를 사용할 때보다 속도가 느려질 수 있습니다.

표시 객체 컨테이너는 다른 표시 객체 컨테이너를 표시 목록에 자식 객체로 포함할 수 있기 때문에 응용 프로그램의 전체 표시 목록을 트리로 탐색할 수 있습니다. 예를 들어, 앞에서 인용한 코드에서 `pict` **Loader** 객체의 로드 작업이 완료되면 `pict` 객체에 비트맵인 자식 표시 객체 하나가 로드됩니다. 이 비트맵 표시 객체에 액세스하려면 `pict.getChildAt(0)`을 입력합니다. 또한 `container.getChildAt(0).getChildAt(0)(container.getChildAt(0) == pict`이기 때문)를 입력할 수도 있습니다.

다음 함수는 표시 객체 컨테이너에서 표시 목록의 `trace()` 출력을 들여쓰기 형식으로 표시합니다.

```
function traceDisplayList(container:DisplayObjectContainer,
                          indentString:String = ""):void
{
  var child:DisplayObject;
  for (var i:uint=0; i < container.numChildren; i++)
  {
    child = container.getChildAt(i);
    trace(indentString, child, child.name);
    if (container.getChildAt(i) is DisplayObjectContainer)
    {
      traceDisplayList(DisplayObjectContainer(child), indentString + "")
    }
  }
}
```

Stage 속성 설정

Stage 클래스는 DisplayObject 클래스의 대부분의 속성과 메서드를 무시합니다. 이 무시되는 속성 또는 메서드 중 하나를 호출하면 Flash Player에서 예외가 발생합니다. 예를 들어, Stage 객체는 응용 프로그램에 대한 기본 컨테이너로 위치가 고정되기 때문에 `x` 또는 `y` 속성이 없습니다. `x` 및 `y` 속성은 컨테이너를 기준으로 표시 객체의 위치를 나타냅니다. Stage는 다른 표시 객체 컨테이너에 포함되지 않기 때문에 이러한 속성이 적용되지 않습니다.

예외	Stage 클래스의 일부 속성과 메서드는 로드된 첫 번째 SWF 파일과 동일한 보안 샌드박스에 없는 표시 객체에는 사용할 수 없습니다. 자세한 내용은 730페이지의 "스테이지 보안"을 참조하십시오.

재생 프레임 속도 제어

Stage 클래스의 `framerate` 속성은 응용 프로그램에 로드된 모든 SWF 파일의 프레임 속도를 설정하는 데 사용됩니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

스테이지 크기 조절 제어

Flash Player 화면 크기를 조절하면 스테이지 내용이 이에 맞게 자동으로 조절됩니다. Stage 클래스의 `scaleMode` 속성은 스테이지 내용 조절 방식을 결정합니다. 이 속성은 flash.display.StageScaleMode 클래스의 상수로 정의되는 서로 다른 네 가지 값으로 설정할 수 있습니다.

세 가지 `scaleMode` 값(`StageScaleMode.EXACT_FIT`, `StageScaleMode.SHOW_ALL` 및 `StageScaleMode.NO_BORDER`)의 경우 Flash Player는 해당 경계에 맞춰 스테이지 내용의 크기를 조절합니다. 이 세 가지 옵션에 따라 다음과 같이 크기 조절 수행 방식이 달라집니다.

- `StageScaleMode.EXACT_FIT` - SWF 크기를 비례적으로 조절합니다.
- `StageScaleMode.SHOW_ALL` - 표준 TV에서 와이드 스크린 영화를 볼 때 나타나는 검정색 바 같은 테두리의 표시 여부를 결정합니다.
- `StageScaleMode.NO_BORDER` - 내용을 부분적으로 자를 수 있는지 여부를 결정합니다.

또는 `scaleMode`가 `StageScaleMode.NO_SCALE`로 설정된 경우, 뷰어에서 Flash Player 윈도우 크기를 조절할 때 스테이지 내용은 각각 정의된 크기로 유지됩니다. 이 크기 조절 모드에서만 Stage 클래스의 `width` 및 `height` 속성을 사용하여 크기 조절된 Flash Player 윈도우의 실제 픽셀 크기를 결정할 수 있습니다. 다른 크기 조절 모드에서는 `stageWidth` 및 `stageHeight` 속성이 항상 SWF의 원래 폭 및 높이를 반영합니다. 또한 `scaleMode`가 `StageScaleMode.NO_SCALE`로 설정되고 SWF 파일의 크기가 조절된 경우 Stage 클래스의 `resize` 이벤트가 전달되어 이에 따라 조절할 수 있습니다.

따라서 scaleMode를 StageScaleMode.NO_SCALE로 설정하면 원하는 경우 윈도우 크기 조절에 맞춰 화면 내용을 조절하는 방법을 보다 세밀하게 제어할 수 있습니다. 예를 들어, 비디오와 컨트롤 막대가 포함된 SWF에서 스테이지의 크기를 조절할 때 컨트롤 막대를 동일한 크기로 유지하고 스테이지 크기 변경에 맞춰 비디오 윈도우의 크기만 변경할 수 있습니다. 다음 예제에서 이를 확인할 수 있습니다.

```
// videoScreen은 비디오를 포함하는 표시 객체(예: Video 인스턴스)로서
// 스테이지의 왼쪽 위 모서리에 배치되어야 하며 SWF 크기 조절 시 videoScreen의
// 크기도 조절되어야 합니다.

// controlBar는 여러 버튼을 포함하는 표시 객체(예: Sprite)로서
// 스테이지의 왼쪽 아래 모서리(videoScreen 아래)에 배치되어야 하며
// SWF 크기 조절 시 controlBar의 크기도 조절되면 안됩니다.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
  var swfWidth:int = swfStage.stageWidth;
  var swfHeight:int = swfStage.stageHeight;

  // 비디오의 윈도우 크기를 조절합니다.
  var newVideoHeight:Number = swfHeight - controlBar.height;
  videoScreen.height = newVideoHeight;
  videoScreen.scaleX = videoScreen.scaleY;

  // 컨트롤 막대의 위치를 변경합니다.
  controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

전체 화면 모드 작업

전체 화면 모드를 사용하면 테두리, 메뉴 모음 등을 사용하지 않고 SWF로 뷰어의 전체 모니터를 채울 수 있습니다. Stage 클래스의 displayState 속성은 SWF에 대한 전체 화면 모드를 설정 및 해제하는 데 사용됩니다. displayState 속성은 flash.display.StageDisplayState 클래스에 상수로 정의된 값 중 하나로 설정할 수 있습니다. 전체 화면 모드를 설정하려면 displayState를 StageDisplayState.FULL_SCREEN으로 설정합니다.

```
// mySprite는 표시 목록에 이미 추가된 Sprite 인스턴스입니다.
mySprite.stage.displayState = StageDisplayState.FULL_SCREEN;
```

전체 화면 모드를 종료하려면 displayState 속성을 StageDisplayState.NORMAL로 설정합니다.

```
mySprite.stage.displayState = StageDisplayState.NORMAL;
```

또한 사용자는 포커스를 다른 윈도우로 전환하거나 Esc 키(모든 플랫폼), Ctrl+W(Windows), Command+W(Mac), Alt+F4(Windows) 등과 같은 여러 키 조합 중 하나를 사용하여 전체 화면 모드를 해제할 수 있습니다.

전체 화면 모드에 대한 Stage 크기 조절 비헤이비어는 일반 모드와 동일합니다. 즉, Stage 클래스의 scaleMode 속성에 의해 크기 조절이 제어됩니다. scaleMode 속성을 StageScaleMode.NO_SCALE로 설정하면 SWF에서 차지하는 화면의 크기(이 경우 전체 화면)에 따라 Stage의 stageWidth 및 stageHeight 속성이 변경됩니다.

Stage 클래스의 fullScreen 이벤트를 사용하여 전체 화면 모드가 설정 또는 해제될 때를 감지하여 응답할 수 있습니다. 예를 들어, 다음 예제처럼 전체 화면 모드를 시작하거나 해제할 때 화면에서 항목을 재배치하거나, 추가 또는 제거할 수 있습니다.

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
  if (event.fullScreen)
  {
    // 입력 텍스트 필드를 제거합니다.
    // 전체 화면 모드를 닫는 버튼을 추가합니다.
  }
  else
  {
    // 입력 텍스트 필드를 다시 추가합니다.
    // 전체 화면 모드를 닫는 버튼을 제거합니다.
  }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN,
  fullScreenRedraw);
```

이 코드에 표시된 것처럼 fullScreen 이벤트에 대한 event 객체는 flash.events.FullScreenEvent 클래스의 인스턴스이며, 전체 화면 모드가 활성화되는지(true) 또는 비활성화되는지(false)를 나타내는 fullScreen 속성을 포함합니다.

ActionScript에서 전체 화면 모드로 작업할 경우 다음 사항을 고려합니다.

- 전체 화면 모드는 ActionScript에서 마우스 클릭(오른쪽 버튼 클릭 포함) 또는 키 입력을 통해서만 시작할 수 있습니다.

- 여러 모니터를 사용하는 사용자의 경우 SWF 내용을 확장하면 모니터 하나만 채워집니다. Flash Player에서는 메트릭을 사용하여 SWF의 가장 많은 부분이 포함된 모니터를 확인하고, 해당 모니터에 전체 화면 모드를 적용합니다.
- HTML 페이지에 포함된 SWF 파일의 경우, 다음과 같이 Flash Player를 포함할 HTML 코드는 이름이 `allowFullScreen`이고 값이 `true`인 `param` 태그와 `embed` 특성을 포함해야 합니다.

```
<object>
  ...
  <param name="allowFullScreen" value="true" />
  <embed ... allowfullscreen="true" />
</object>
```

  웹 페이지에서 JavaScript를 사용하여 SWF 포함 태그를 생성할 경우 `allowFullScreen` `param` 태그 및 특성을 추가하도록 JavaScript를 변경해야 합니다. 예를 들어, HTML 페이지에서 `AC_FL_RunContent()` 함수(Flex Builder와 Flash 생성 HTML 페이지 모두에 사용됨)를 사용하는 경우, 다음과 같이 `allowFullScreen` 매개 변수를 해당 함수 호출에 추가해야 합니다.

```
AC_FL_RunContent(
  ...
  'allowFullScreen','true',
  ...
  ); //AC code 끝
```

  독립 실행형 Flash Player에서 실행 중인 SWF 파일에는 적용되지 않습니다.
- 키보드 이벤트 및 TextFields 인스턴스의 텍스트 입력 등과 같은 모든 키보드 관련 ActionScript는 전체 화면 모드에서 비활성화됩니다. 전체 화면 모드를 닫는 키보드 단축키는 예외입니다.

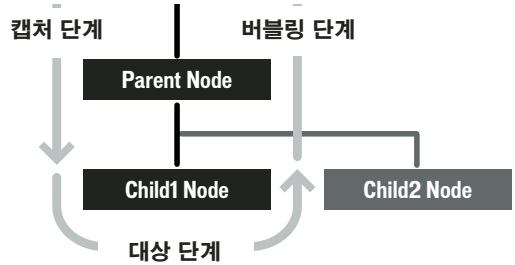
또한 이해해야 할 몇 가지 추가 보안 관련 제한 사항이 있습니다. 이러한 제한 사항에 대해서는 719페이지의 “보안 샌드박스”에 설명되어 있습니다.

표시 객체에 대한 이벤트 처리

DisplayObject 클래스는 EventDispatcher 클래스에서 상속됩니다. 즉, 모든 표시 객체는 이벤트 모델에 모두 사용할 수 있습니다(295페이지의 제10장, “이벤트 처리” 참조). 모든 표시 객체는 EventDispatcher 클래스에서 상속되는 `addEventListener()` 메서드를 사용하여 특정 이벤트를 수신할 수 있습니다. 이때 수신 객체가 해당 이벤트에 대한 이벤트 흐름의 일부여야 합니다.

Flash Player에서 이벤트 객체를 전달할 경우 해당 이벤트 객체는 Stage부터 이벤트가 발생한 표시 객체까지를 왕복합니다. 예를 들어, `child1`이라는 표시 객체를 클릭하면 Flash Player는 Stage에서 표시 목록 계층 아래의 `child1` 표시 객체까지 이벤트 객체를 전달합니다.

다음 다이어그램에 표시된 것처럼 이벤트 흐름은 개념적으로 3단계로 구분됩니다.



자세한 내용은 295페이지의 제10장, “이벤트 처리”를 참조하십시오.

표시 객체 이벤트 작업을 할 경우 주의해야 할 중요 사항은, 표시 객체를 표시 목록에서 제거할 때 표시 객체가 메모리에서 자동으로 제거(가비지 컬렉션)되는지 여부에 대해 이벤트 리스너가 미치는 영향입니다. 표시 객체에 이벤트에 대한 리스너로 구독되는 객체가 있는 경우 해당 표시 객체는 표시 목록에서 제거되더라도 해당 리스너 객체에 대한 참조가 여전히 존재하기 때문에 메모리에서 제거되지 않습니다. 자세한 내용은 312페이지의 “이벤트 리스너 관리”를 참조하십시오.

DisplayObject 하위 클래스 선택

선택할 다양한 옵션이 있는 표시 객체로 작업할 경우 사용할 표시 객체와 용도를 결정해야 합니다. 다음과 같이 결정에 도움이 되는 몇 가지 지침이 있습니다. 클래스의 인스턴스가 필요한 경우나 만들 클래스에 대한 기본 클래스를 선택하는 경우에도 동일한 제안 사항이 적용됩니다.

- 다른 표시 객체의 컨테이너로 사용할 객체가 필요하지 않는 경우(즉, 독립 실행형 화면 요소 역할을 하는 객체만 필요한 경우) 용도에 따라 DisplayObject 또는 InteractiveObject 하위 클래스 중 하나를 선택합니다.
  - 비트맵 이미지 표시를 위한 Bitmap
  - 텍스트 추가를 위한 TextField
  - 비디오 표시를 위한 Video
  - 화면 상에 내용을 그리기 위한 “캔바스” Shape. 특히, 화면에서 모양을 그리기 위한 인스턴스를 만들 경우 해당 인스턴스가 다른 표시 객체의 컨테이너가 아니라면, Sprite 또는 MovieClip 대신 Shape를 사용하면 성능이 크게 향상됩니다.

- Flash 제작용 MorphShape, StaticText 또는 SimpleButton 항목. 이러한 클래스의 인스턴스를 프로그램 방식으로 만들 수는 없지만, Flash 제작 프로그램을 사용하여 만든 항목을 나타내도록 이러한 데이터 유형으로 변수를 만들 수 있습니다.

- 변수가 기본 Stage를 나타내야 하는 경우 Stage 클래스를 데이터 유형으로 사용합니다.
- 외부 SWF 파일 또는 이미지 파일을 로드하기 위한 컨테이너가 필요한 경우 Loader 인스턴스를 사용합니다. 로드된 내용은 표시 목록에 Loader 인스턴스의 자식으로 추가됩니다. 데이터 유형은 로드된 내용의 특성에 따라 다음과 같이 달라집니다.
  - 로드된 이미지는 Bitmap 인스턴스입니다.
  - ActionScript 3.0에서 작성된 로드된 SWF 파일은 Sprite 또는 MovieClip 인스턴스이거나 내용 작성자에 의해 지정된 해당 클래스의 하위 클래스 인스턴스가 됩니다.
  - ActionScript 1.0 또는 ActionScript 2.0에서 작성된 로드된 SWF 파일은 AVM1Movie 인스턴스가 됩니다.
- ActionScript를 사용하여 표시 객체에 그릴지 여부에 관계없이 다른 표시 객체에 대한 컨테이너 역할을 할 객체가 필요한 경우 DisplayObjectContainer 하위 클래스 중 하나를 선택합니다.
  - ActionScript만 사용하여 객체를 만들거나 ActionScript만 사용하여 만들고 조작할 사용자 정의 표시 객체에 대한 기본 클래스로 객체를 만드는 경우 Sprite를 선택합니다.
  - Flash 제작 도구에서 만든 무비 클립 심볼을 가리키는 변수를 만들 경우 MovieClip을 선택합니다.
- Flash 라이브러리에서 무비 클립 심볼과 연결될 클래스를 만들 경우 DisplayObjectContainer 하위 클래스 중 하나를 클래스의 기본 클래스로 선택합니다.
  - 관련된 무비 클립 심볼의 여러 프레임에 내용이 있는 경우 MovieClip을 선택합니다.
  - 관련된 무비 클립 심볼의 첫 번째 프레임에만 내용이 있는 경우 Sprite를 선택합니다.

표시 객체 조작

사용하려고 선택한 표시 객체에 관계없이 모든 표시 객체에는 화면에 표시되는 요소로서의 공통적인 다양한 조작 방법이 있습니다. 예를 들어, 모든 표시 객체를 화면에 배치한 다음 표시 객체가 쌓이는 순서를 앞 또는 뒤로 이동하거나, 표시 객체의 크기를 조절하거나 회전할 수 있습니다. 모든 표시 객체는 공통 기본 클래스(DisplayObject)로부터 이 기능을 상속하기 때문에 TextField 인스턴스, Video 인스턴스, Shape 인스턴스 또는 다른 표시 객체를 조작하든 관계없이 이 기능은 동일하게 동작합니다. 다음 단원에서는 이러한 공통 표시 객체 조작 방법 중 몇 가지를 자세히 설명합니다.

위치 변경

화면에 표시 객체를 배치하는 것은 표시 객체에 대한 가장 기본적인 조작입니다. 표시 객체의 위치를 설정하려면 객체의 `x` 및 `y` 속성을 변경합니다.

```
myShape.x = 17;
myShape.y = 212;
```

표시 객체 위치 지정 시스템에서는 스테이지를 직교 좌표계(가로 x축과 세로 y축이 있는 일반 격자 시스템)로 취급합니다. 좌표계의 원점(x축과 y축이 만나는 0,0 좌표)은 Stage의 왼쪽 위 모서리에 있습니다. x 값은 원점으로부터 오른쪽이 양수이고 왼쪽이 음수이지만, y 값은 아래쪽이 양수이고 위쪽이 음수로 일반 그래프 시스템과 반대입니다. 예를 들어, 이전 코드 행은 `myShape` 객체를 x 좌표 17(원점의 오른쪽으로 17픽셀) 및 y 좌표 212(원점의 아래로 212픽셀)로 이동합니다.

기본적으로 ActionScript를 사용하여 표시 객체를 만들 경우 `x` 속성과 `y` 속성이 모두 0으로 설정되어 객체가 부모 내용의 왼쪽 위 모서리에 배치됩니다.

스테이지를 기준으로 위치 변경

`x` 및 `y` 속성은 항상 부모 표시 객체 축의 0,0 좌표를 기준으로 표시 객체의 위치를 나타냅니다. Sprite 인스턴스에 포함된 Shape 인스턴스(예: circle)에 대해 Shape 객체의 `x` 및 `y` 속성을 0으로 설정하면 원이 Sprite의 왼쪽 위 모서리에 배치됩니다. 이 위치가 반드시 Stage의 왼쪽 위 모서리가 되는 것는 아닙니다. 전역 스테이지 좌표를 기준으로 객체를 배치하려면 표시 객체의 `globalToLocal()` 메서드를 사용하여 좌표를 전역(스테이지) 좌표에서 로컬(표시 객체 컨테이너) 좌표로 다음과 같이 변환할 수 있습니다.

```
// 부모의 위치에 관계없이 스테이지의 왼쪽 위 모서리에
// 모양을 배치합니다 .

// 위치가 x:200 및 y:200 인 Sprite 를 만듭니다 .
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
```

```
this.addChild(mySprite);

// Sprite의 0,0 좌표에 점을 참조용으로 그립니다.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// 원 Shape 인스턴스를 만듭니다.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Shape에 반경이 50이며 중심점이 x:50, y:50에 있는 원을 그립니다.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Shape의 왼쪽 위 모서리가 스테이지의 0, 0 좌표에 위치하도록 Shape를 이동합니다.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

마찬가지로 DisplayObject 클래스의 `localToGlobal()` 메서드를 사용하여 로컬 좌표를 스테이지 좌표로 변환할 수 있습니다.

드래그 앤 드롭 상호 작용 만들기

일반적으로 표시 객체를 이동하는 한 가지 이유는 사용자가 객체를 클릭한 상태로 마우스를 이동하면 마우스 버튼을 놓을 때까지는 객체가 따라서 이동하도록 드래그 앤 드롭 상호 작용을 만들기 위한 것입니다. 드래그 앤 드롭 상호 작용은 ActionScript에서 두 가지 방법으로 만들 수 있습니다. 두 방법 모두 두 마우스 이벤트가 사용됩니다. 즉, 마우스 버튼을 누르면 마우스 커서를 따라 이동하도록 객체에게 지시하고, 마우스 버튼을 놓으면 마우스 커서를 따라 이동하는 것을 중지하도록 객체에게 지시합니다.

`startDrag()` 메서드를 사용하는 첫 번째 방법은 더 간단하지만 보다 제한적입니다. 마우스 버튼을 누르면 드래그할 표시 객체의 `startDrag()` 메서드가 호출됩니다. 마우스 버튼을 놓으면 `stopDrag()` 메서드가 호출됩니다.

```
// 이 코드는 startDrag() 기술을 사용하여 드래그 앤 드롭 상호 작용을 만듭니다.
// 사각형은 DisplayObject(예: MovieClip 또는 Sprite 인스턴스)입니다.

import flash.events.MouseEvent;

// 마우스 버튼을 누르면 이 함수가 호출됩니다.
```

```
function startDragging(event:MouseEvent):void
{
  square.startDrag();
}

// 마우스 버튼을 놓으면 이 함수가 호출됩니다 .
function stopDragging(event:MouseEvent):void
{
  square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

이 기술은 `startDrag()`를 사용하여 한 번에 하나의 항목만 드래그할 수 있다는 한 가지 중요한 제한이 있습니다. 표시 객체를 드래그하는 중에 `startDrag()` 메서드가 다른 표시 객체에서 호출되면, 마우스를 따라 이동하던 첫 번째 표시 객체의 이동이 즉시 중지됩니다. 예를 들어, `startDragging()` 함수가 아래에 표시된 것처럼 변경되면 `square.startDrag()` 메서드를 호출하더라도 `circle` 객체만 드래그됩니다.

```
function startDragging(event:MouseEvent):void
{
  square.startDrag();
  circle.startDrag();
}
```

`startDrag()`를 사용하여 객체를 한 번에 하나씩만 드래그할 수 있기 때문에, 임의의 표시 객체에 대해 `stopDrag()` 메서드를 호출하여 현재 드래그 중인 객체의 이동을 중지시킬 수 있습니다.

여러 표시 객체를 드래그해야 하거나 여러 객체가 `startDrag()`를 사용하여 발생할 수 있는 충돌을 방지하려면 mouse-following 기술을 사용하여 드래그 효과를 만드는 것이 좋습니다. 이 기술을 사용할 경우 마우스 버튼을 누르면 함수가 Stage의 `mouseMove` 이벤트에 대한 리스너로 구독됩니다. 그러면 마우스를 이동할 때마다 이 함수가 호출되어 드래그되는 객체가 마우스의 x, y 좌표로 바로 이동됩니다. 마우스 버튼을 놓으면 함수가 구독 해제되어, 마우스를 이동해도 함수가 더 이상 호출되지 않으므로 객체가 마우스 커서를 따라 이동하지 않습니다. 다음은 이 기술을 설명하는 일부 코드입니다.

```
// 이 코드는 mouse-following 기술을 사용하여 드래그 앤 드롭 상호 작용을 만듭니다 .
// 원은 DisplayObject( 예 : MovieClip 또는 Sprite 인스턴스 ) 입니다 .

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// 마우스 버튼을 누르면 이 함수가 호출됩니다 .
function startDragging(event:MouseEvent):void
{
```

```
    // 마우스 버튼을 눌렀을 때 커서 위치와 마우스
    // 버튼을 놓았을 때 원의 x, y 좌표 간의 차이(오프셋)를
    // 기록합니다.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // mouseMove 이벤트 수신을 시작하도록 Flash Player에 지시합니다.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// 마우스 버튼을 놓으면 이 함수가 호출됩니다.
function stopDragging(event:MouseEvent):void
{
    // mouseMove 이벤트 수신을 중지하도록 Flash Player에 지시합니다.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// 마우스 버튼을 누른 상태에서 마우스를 움직일 때마다 이 함수가 호출됩니다.
function dragCircle(event:MouseEvent):void
{
    // 커서 위치와 드래그된 객체 위치 간의 오프셋을 유지하면서 커서 위치로
    // 원을 이동합니다.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // 이 이벤트 후 화면을 새로 고치도록 Flash Player에 지시합니다.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

표시 객체를 마우스 커서를 따라 이동하게 하는 것 외에도, 드래그 앤 드롭 상호 작용은 드래그된 객체를 표시의 전면으로 이동시켜 모든 다른 객체 위에 떠 있는 것처럼 보이게 할 수 있습니다. 예를 들어, 드래그 앤 드롭 상호 작용을 하는 원과 사각형의 두 객체가 있다고 가정합니다. 원이 표시 목록에서 사각형의 아래에 있을 때 원을 클릭한 상태에서 드래그하여 커서를 사각형 위에 놓으면, 원이 사각형 뒤로 서서히 이동하는 것처럼 표시되어 드래그 앤 드롭 효과가 제거됩니다. 대신 원을 클릭하면 원이 표시 목록의 맨 위로 이동하여 항상 다른 내용의 위에 표시되게 할 수 있습니다.

이전 예제에서 사용된 다음 코드는 두 표시 객체인 원과 사각형에 대한 드래그 앤 드롭 상호 작용을 만듭니다. 어느 한 객체를 마우스 버튼으로 누르면 해당 항목이 Stage 표시 목록의 위로 이동하여 드래그된 항목이 항상 위에 표시됩니다. 새 코드나 이전 샘플에서 변경된 코드는 굵은 글꼴로 표시됩니다.

```
// 이 코드는 mouse-following 기술을 사용하여 드래그 앤 드롭 상호 작용을 만듭니다.
// 원과 사각형은 DisplayObjects(예: MovieClip 또는 Sprite 인스턴스)입니다.

import flash.display.DisplayObject;
```

```
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// 마우스 버튼을 누르면 이 함수가 호출됩니다.
function startDragging(event:MouseEvent):void
{
  // 드래그하고 있는 객체를 기억합니다.
  draggedObject = DisplayObject(event.target);

  // 마우스 버튼을 눌렀을 때 커서 위치와 마우스 버튼을 놓았을 때 드래그된 객체의
  // x, y 좌표 간의 차이(오프셋)를 기록합니다.
  offsetX = event.stageX - draggedObject.x;
  offsetY = event.stageY - draggedObject.y;

  // 선택한 객체를 표시 목록의 맨 위로 이동합니다.
  stage.addChild(draggedObject);

  // TmouseMove 이벤트 수신을 시작하도록 Flash Player에 지시합니다.
  stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// 마우스 버튼을 놓으면 이 함수가 호출됩니다.
function stopDragging(event:MouseEvent):void
{
  // mouseMove 이벤트 수신을 중지하도록 Flash Player에 지시합니다.
  stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// 마우스 버튼을 누른 상태에서 마우스를 움직일 때마다 이 함수가 호출됩니다.
function dragObject(event:MouseEvent):void
{
  // 커서 위치와 드래그된 객체 위치 간의 오프셋을 유지하면서 커서 위치로
  // 드래그된 객체를 이동합니다.
  draggedObject.x = event.stageX - offsetX;
  draggedObject.y = event.stageY - offsetY;

  // 이 이벤트 후 화면을 새로 고치도록 Flash Player에 지시합니다.
  event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

토큰이나 카드가 더미 간에 이동하는 게임에서처럼 이 효과를 더 확장하려면, 객체를 "선택"하여 드래그된 객체를 스테이지의 표시 목록에 추가한 다음, 마우스 버튼을 놓아 해당 객체를 다른 표시 목록(예: 드롭되는 "더미")에 추가할 수 있습니다.

마지막으로 효과를 향상시키기 위해 클릭할 때(드래그를 시작할 때) 표시 객체에 그림자 필터를 적용하고 객체를 놓을 때 그림자를 제거할 수 있습니다. ActionScript에서 그림자 필터 및 기타 표시 객체 필터를 사용하는 방법에 대한 자세한 내용은 437페이지의 제15장, "표시 객체 필터링"을 참조하십시오.

표시 객체 패닝 및 스크롤

표시 객체가 표시하려는 영역에 비해 너무 큰 경우 `scrollRect` 속성을 사용하여 표시 객체의 표시 가능한 영역을 정의할 수 있습니다. 또한 사용자 입력에 따라 `scrollRect` 속성을 변경하여 내용을 왼쪽/오른쪽으로 패닝하거나 위/아래로 스크롤할 수 있습니다.

`scrollRect` 속성은 사각형 영역을 단일 객체로 정의하는 데 필요한 값을 결합하는 클래스인 Rectangle 클래스의 인스턴스입니다. 표시 객체의 표시 가능한 영역을 처음으로 정의하려면 새 Rectangle 인스턴스를 만들어 표시 객체의 `scrollRect` 속성에 할당합니다. 나중에 스크롤 또는 패닝하려면 `scrollRect` 속성을 개별 Rectangle 변수로 읽고 원하는 속성을 변경합니다. 예를 들어, Rectangle 인스턴스의 `x` 속성을 변경하여 패닝하거나 `y` 속성을 변경하여 스크롤합니다. 그런 다음 Rectangle 인스턴스를 `scrollRect` 속성에 다시 할당하여 표시 객체에 변경된 값을 알립니다.

예를 들어, 다음 코드에서는 너무 커서 SWF 파일의 경계 안에 들어가지 않는 `bigText`라는 TextField 객체에 대한 표시 가능한 영역을 정의합니다. 두 버튼 `up` 및 `down`을 클릭하면 `scrollRect` Rectangle 인스턴스의 `y` 속성을 수정하여 TextField 객체의 내용을 위 또는 아래로 스크롤하는 함수가 호출됩니다.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// TextField 인스턴스의 표시 가능한 영역을 처음으로 정의합니다.
// 왼쪽: 0, 위쪽: 0, 폭: TextField, 높이: 350 픽셀
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// TextField를 비트맵으로 캐시하여 성능을 향상시킵니다.
bigText.cacheAsBitmap = true;

// 버튼을 클릭할 때 호출됩니다.
function scrollUp(event:MouseEvent):void
{
  // 현재 스크롤 사각형에 액세스합니다.
  var rect:Rectangle = bigText.scrollRect;
  // 사각형을 20픽셀 아래로 이동하면서 사각형의 y 값을 20만큼 줄입니다.
  rect.y -= 20;
  // TextField에 사각형을 다시 지정하여 변경 내용을 적용합니다.
```

```
    bigText.scrollRect = rect;
}

// "down" 버튼을 클릭할 때 호출됩니다 .
function scrollDown(event:MouseEvent):void
{
    // 현재 스크롤 사각형에 액세스합니다 .
    var rect:Rectangle = bigText.scrollRect;
    // 사각형을 20 픽셀 위로 이동하면서 사각형의 y 값을 20 만큼 늘립니다 .
    rect.y += 20;
    // TextField 에 사각형을 다시 지정하여 변경 내용을 적용 ii 합니다 .
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

이 예제에서 보듯이 표시 객체의 `scrollRect` 속성으로 작업할 경우 `cacheAsBitmap` 속성을 사용하여 Flash Player가 표시 객체의 내용을 비트맵으로 캐시하도록 지정하는 것이 좋습니다. 그렇게 하면 표시 객체의 내용을 스크롤할 때마다 Flash Player에서 표시 객체의 전체 내용을 다시 그릴 필요가 없을 뿐만 아니라, 캐시된 비트맵을 사용하여 필요한 부분을 화면에 직접 렌더링할 수 있습니다. 자세한 내용은 381페이지의 "표시 객체 캐싱"을 참조하십시오.

객체 크기 조작 및 크기 조절

크기 속성(`width` 및 `height`) 또는 크기 조절 속성(`scaleX` 및 `scaleY`)을 사용하여 표시 객체의 크기를 두 가지 방식으로 측정하고 조작할 수 있습니다.

모든 표시 객체에는 `width` 속성과 `height` 속성이 있습니다. 이러한 속성은 처음에는 객체의 크기(픽셀)로 설정됩니다. 이러한 속성의 값을 읽어서 표시 객체의 크기를 측정할 수 있습니다. 다음과 같이 새 값을 지정하여 객체의 크기를 변경할 수도 있습니다.

```
// 표시 객체의 크기를 조절합니다 .
square.width = 420;
square.height = 420;

// 원 표시 객체의 반경을 결정합니다 .
var radius:Number = circle.width / 2;
```

표시 객체의 `height` 또는 `width`를 변경하면 객체의 크기가 조절됩니다. 즉, 객체의 내용이 새 영역에 맞게 확대되거나 축소됩니다. 표시 객체에 벡터 모양만 포함되어 있는 경우 품질 손상 없이 해당 모양이 새 크기로 다시 그려집니다. 표시 객체에 있는 비트맵 그래픽 요소는 다시 그려지지 않고 크기가 조절됩니다. 예를 들어, 폭과 높이가 이미지의 실제 픽셀 정보 크기보다 크게 확대되는 디지털 사진은 픽셀화되어 들쭉날쭉하게 보입니다.

표시 객체의 width 또는 height 속성을 변경하면 Flash Player에서 객체의 scaleX 및 scaleY 속성도 함께 업데이트합니다. 이러한 속성은 원본 크기와 비교한 표시 객체의 상대적인 크기를 나타냅니다. scaleX 및 scaleY 속성은 분수(소수) 값을 사용하여 백분율을 나타냅니다. 예를 들어, 표시 객체의 width 속성이 원본 폭의 1/2 크기로 변경된 경우 객체의 scaleX 속성 값은 .5(50%)가 됩니다. 높이가 두 배로 커지면 scaleY 속성 값이 2(200%)가 됩니다.

```
// 원은 폭과 높이가 각각 150 픽셀인 표시 객체입니다 .
// 원래 크기에서 scaleX 와 scaleY 는 1(100%) 입니다 .
trace(circle.scaleX);// 출력 : 1
trace(circle.scaleY); // 출력 : 1

// width 및 height 속성을 변경하면 그에 따라 scaleX 및 scaleY 속성도 변경됩니다 .
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // 출력 : 0.6622516556291391
trace(circle.scaleY); // 출력 : 0.4966887417218543
```

크기는 비례적으로 변경되지 않습니다. 즉, 정사각형의 height 속성만 변경하고 width 속성은 변경하지 않을 경우 가로와 세로의 비율이 달라져서 정사각형이 아니라 직사각형이 됩니다. 표시 객체의 크기를 상대적으로 변경하려면 width 또는 height 속성 대신 scaleX 및 scaleY 속성 값을 설정하여 객체의 크기를 조절할 수 있습니다. 예를 들어, 다음 코드는 square라는 표시 객체의 width 속성을 변경한 다음 세로 비율(scaleY)을 가로 비율과 일치하도록 변경하여 정사각형의 가로 세로 비율을 유지합니다.

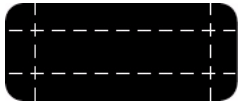
```
// 폭을 직접 변경합니다 .
square.width = 150;

// 가로 비율에 맞게 세로 비율을 변경하여 가로 세로 비율을 유지합니다 .
square.scaleY = square.scaleX;
```

크기 조절 시 왜곡 제어

일반적으로 표시 객체의 크기를 조절하면(예: 가로로 확대) 각 부분이 동일한 양만큼 확장되어 객체 전체에서 균일하게 왜곡이 발생합니다. 그래픽 및 디자인 요소의 경우 이런 결과를 원할 수도 있습니다. 그러나, 표시 객체의 확장되는 부분과 변경되지 않는 부분을 제어해야 할 경우도 있습니다. 그러한 일반적인 예로 모서리가 둥근 사각형 버튼이 있습니다. 일반 크기 조절에서는 버튼의 모서리가 확장되어 버튼 크기를 변경하면 모서리 반경도 변경됩니다.

그러나, 이 경우에는 왜곡 현상 없이 크기가 조절되도록 크기 조절을 제어해야 합니다.
즉, 크기를 조절할 영역(직선 변 및 중간)과 크기를 조절하지 않을 영역(모서리)을 지정합니다.

9 슬라이스 크기 조절(Scale-9)을 사용하여 객체 크기 조절 방법을 제어할 표시 객체를 만들 수 있습니다. 9 슬라이스 크기 조절을 사용하면 표시 객체가 9개의 개별 사각형(3 x 3 격자, tic-tac-toe 보드의 격자와 비슷)으로 나뉘어집니다. 각 사각형의 크기가 반드시 동일할 필요는 없습니다. 사용자가 격자선의 위치를 지정합니다. 버튼의 둥근 모서리처럼 네 모서리의 사각형에 놓이는 내용은 표시 객체의 크기를 조절해도 확장되거나 축소되지 않습니다. 위쪽 가운데 사각형과 아래쪽 가운데 사각형은 가로로 크기가 조절되고 세로는 변경되지 않는 반면에, 왼쪽 중앙 및 오른쪽 중앙 사각형은 세로로 크기가 조절되고 가로는 변경되지 않습니다. 가운데 사각형은 가로와 세로 방향 모두 크기가 조절됩니다.



표시 객체를 만들 때 특정 내용의 크기가 조절되지 않게 하려면 해당 내용이 모서리 사각형 중 하나에서 끝나도록 9 슬라이스 크기 조절 격자의 구분선을 배치해야 합니다.

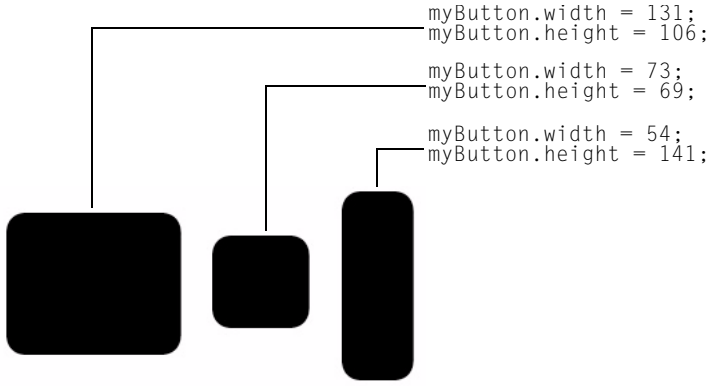
ActionScript에서 표시 객체의 `scale9Grid` 속성 값을 설정하여 객체에 대한 9 슬라이스 크기 조절을 지정하고 객체의 Scale-9 격자로 사각형 크기를 정의합니다. 다음과 같이 Rectangle 클래스의 인스턴스를 `scale9Grid` 속성 값으로 사용할 수 있습니다.

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Rectangle 생성자의 네 매개 변수는 x 좌표, y 좌표, 폭 및 높이입니다. 이 예제에서 사각형의 왼쪽 위 모서리는 `myButton` 표시 객체에서 x: 32, y: 27 지점에 위치합니다. 사각형은 폭이 71픽셀이고 높이가 65픽셀입니다. 따라서 사각형의 오른쪽 가장자리는 표시 객체의 x 좌표 103 위치에 있고 아래쪽 가장자리는 표시 객체의 y 좌표 92 위치에 있습니다.

Rectangle 인스턴스에 의해 정의된 영역에 포함된 실제 영역은 Scale-9 격자의 가운데 사각형을 나타냅니다. 다음 그림에 표시된 것처럼 Flash Player에서 Rectangle 인스턴스의 변을 확장하여 다른 사각형을 계산합니다.

이 경우 버튼 크기를 확대하거나 축소하면 둥근 모서리는 확장되거나 축소되지 않지만, 다른 영역은 해당 크기 조절에 따라 조절됩니다.



표시 객체 캐싱

응용 프로그램을 작성하든 스크립팅된 복잡한 애니메이션을 작성하든 Flash에서 작업하는 디자인은 규모가 커질 수 있으므로, 성능 및 최적화를 고려해야 합니다. 사각형 Shape 인스턴스와 같이 정적으로 유지되는 내용은 자동으로 최적화되지 않습니다. 따라서, 사각형의 위치를 변경할 경우 Flash는 전체 Shape 인스턴스를 다시 그립니다.

지정된 표시 객체를 캐시하여 SWF 파일의 성능을 향상시킬 수 있습니다. 표시 객체는 기본적으로 인스턴스 벡터 데이터의 비트맵 버전인 *표면*이며, 이것은 SWF 파일을 만드는 과정에서 자주 변경하지 않는 데이터입니다. 그러므로 캐싱이 활성화된 인스턴스는 SWF 파일이 재생될 때 계속적으로 다시 그려지지 않습니다. 따라서 SWF 파일은 빠르게 렌더링됩니다.

예	**벡터 데이터는 표면이 다시 만들어질 때 업데이트할 수 있습니다. 그러므로 표면에 캐싱된 벡터 데이터는 전체 SWF 파일에서 동일하게 유지될 필요가 없습니다.**

표시 객체의 cacheAsBitmap 속성을 true로 설정하면 표시 객체가 자체 비트맵 표현을 캐시합니다. Flash는 인스턴스에 대해 표면 객체를 만들며 이 객체는 벡터 데이터 대신에 비트맵으로 캐시된 것입니다. 표시 객체의 경계를 변경하면 표면은 크기가 조절되지 않고 다시 만들어집니다. 표면은 서로 다른 표면 안에 중첩이 가능합니다. 이 경우, 자식 표면은 부모 표면으로 비트맵을 복제하게 됩니다. 자세한 내용은 384페이지의 "비트맵 캐싱 활성화"를 참조하십시오.

DisplayObject 클래스의 opaqueBackground 및 scrollRect 속성은 cacheAsBitmap 속성을 사용하여 비트맵 캐싱과 연관됩니다. 위 세 가지 속성은 서로 독립적이지만, opaqueBackground 및 scrollRect 속성은 객체가 비트맵으로 캐시되는 경우 가장 잘 작동합니다. 즉, opaqueBackground 및 scrollRect 속성은 cacheAsBitmap이 true로 설정된 경우에만 성능상의 이점을 제공합니다. 표시 객체 내용 스크롤에 대한 자세한 내용은 377페이지의 "표시 객체 패닝 및 스크롤"을 참조하십시오. 불투명 배경 설정에 대한 자세한 내용은 384페이지의 "불투명한 배경색 설정"을 참조하십시오.

알파 채널 마스킹을 사용하려면 cacheAsBitmap 속성을 true로 설정해야 하며, 자세한 내용은 390페이지의 "알파 채널 마스킹"을 참조하십시오.

캐싱 기능이 필요한 경우

표시 객체에 캐싱을 활성화하면 표면이 만들어지며, 복잡한 벡터 애니메이션을 빠르게 렌더링할 수 있는 등 여러 이점이 있습니다. 캐싱을 활성화해야 할 여러 경우가 있습니다. SWF 파일의 성능을 향상하려면 항상 캐싱을 활성화하는 것이 좋을 것처럼 보이지만, 캐싱을 활성화해도 성능이 향상되지 않거나 오히려 저하되는 경우도 있습니다. 이 단원에서는 캐싱을 사용해야 할 경우와 일반 표시 객체를 사용해야 할 경우에 대해 설명합니다.

캐시된 데이터의 전체적인 성능은 인스턴스 벡터 데이터의 복잡도, 변경하려는 데이터의 양, opaqueBackground 속성의 설정 여부 등에 달려 있습니다. 작은 영역을 변경할 경우 표면과 벡터 데이터 중 어느 것을 사용해도 결과는 비슷합니다. 응용 프로그램을 배포하기 전에 두 가지 경우를 모두 테스트하는 것도 좋습니다.

비트맵 캐싱 기능이 필요한 경우

다음과 같은 경우에는 일반적으로 비트맵 캐싱을 사용할 경우 큰 이점을 얻을 수 있습니다.

- 복잡한 배경 이미지: 섬세하고 복잡한 벡터 데이터 배경 이미지가 포함된 응용 프로그램을 사용할 경우로, 비트맵 추적 명령을 적용한 이미지 또는 Adobe Illustrator®에서 작성한 아트웍 작업 등입니다. 배경 위에 문자 애니메이션 효과를 적용하는 경우, 배경에서 벡터 데이터를 지속적으로 다시 생성해야 하므로 애니메이션의 속도가 느려집니다. 이때 성능을 향상시키려면 배경 표시 객체의 opaqueBackground 속성을 true로 설정합니다. 배경이 비트맵으로 렌더링되고 빠르게 다시 그려지므로 애니메이션은 훨씬 더 빠르게 재생됩니다.

- 스크롤 텍스트 필드: 스크롤 텍스트 필드에 대량의 텍스트를 표시하는 응용 프로그램을 사용할 경우입니다. 스크롤 경계선으로 스크롤이 가능하도록 설정(즉, `scrollRect` 속성을 설정함)한 표시 객체 안에 텍스트 필드를 배치할 수 있습니다. 이렇게 하면 지정된 인스턴스에 대해 빠른 픽셀 스크롤을 수행할 수 있습니다. 사용자가 표시 객체 인스턴스를 스크롤하면 스크롤된 픽셀은 위쪽으로 이동하며 전체 텍스트 필드가 다시 생성되는 대신 새롭게 표시되는 영역만 생성됩니다.
- 윈도우 시스템: 윈도우가 겹쳐지는 복잡한 구조가 포함된 응용 프로그램을 사용할 경우입니다. 각 윈도우는, 예를 들면 웹 브라우저 윈도우처럼 열려져 있거나 닫혀져 있을 수도 있습니다. `cacheAsBitmap` 속성을 `true`로 설정하여 각 윈도우를 표면으로 표시하면 각 윈도우는 서로 분리되고 캐시됩니다. 사용자는 윈도우를 드래그하여 서로 겹쳐지게 할 수 있으며, 각 윈도우에서는 벡터 내용을 다시 생성할 필요가 없습니다.
- 알파 채널 마스킹: 알파 채널 마스킹을 사용할 경우 `chacheAsBitmap` 속성을 `true`로 설정해야 합니다. 자세한 내용은 390페이지의 "알파 채널 마스킹"을 참조하십시오.

이와 같은 모든 경우에 비트맵 캐싱을 활성화하면 벡터 그래픽이 최적화되므로 응용 프로그램의 응답성 및 상호 작용성이 향상됩니다.

또한 표시 객체에 필터를 적용할 때마다 Flash Player에서 `cacheAsBitmap`이 자동으로 `true`로 설정됩니다. 이 속성을 `false`로 명시적으로 설정한 경우에도 마찬가지입니다. 표시 객체에서 모든 필터를 지우면 `cacheAsBitmap` 속성이 마지막 설정된 값으로 돌아갑니다.

비트맵 캐싱의 사용을 피해야 할 경우

이 기능을 잘못 사용하면 SWF 파일에 부정적인 영향을 미칩니다. 비트맵 캐싱을 사용할 경우 다음 지침을 따라야 합니다.

- 표면(캐싱이 활성화된 표시 객체)을 과도하게 사용하지 않습니다. 각 표면은 일반 표시 객체보다 메모리를 더 사용하므로 렌더링 성능을 향상시킬 경우에만 표면을 활성화해야 합니다.

  캐시된 비트맵은 일반 표시 객체보다 훨씬 더 많은 메모리를 사용할 수도 있습니다. 예를 들어, Stage 위의 크기가 250 x 250픽셀인 Sprite 인스턴스가 캐시될 경우 250KB가 사용되는 반면에, 캐시되지 않은 일반 Sprite 인스턴스는 1KB를 사용합니다.
- 캐시된 표면에 확대/축소를 적용하지 않습니다. 비트맵 캐싱을 남용하면 특히 내용을 확대할 경우 상당한 양의 메모리가 소비됩니다(앞의 항목 참조).
- 표면은 대부분 정적인, 즉 움직임이 거의 없는 표시 객체 인스턴스에 대해 사용합니다. 이 경우에 인스턴스를 드래그하거나 이동할 수 있지만 인스턴스의 내용은 움직이거나 크게 변경되어서는 안 됩니다. 애니메이션 또는 변경되는 내용은 애니메이션 또는 Video 인스턴스가 포함된 MovieClip 인스턴스와 비슷합니다. 예를 들어, 인스턴스를 회전하거나 변형하면 해당 인스턴스는 표면 및 벡터 데이터 간에서 변경되는데, 이는 처리하기 어려운 작업이며 SWF 파일에 부정적인 영향을 미치기도 합니다.

- 표면을 벡터 데이터와 혼합하면 Flash Player 또는 컴퓨터에서 수행해야 하는 처리 작업이 증가됩니다. 가능하면 윈도우 응용 프로그램을 작성할 경우처럼 표면을 서로 그룹화하는 것이 좋습니다.

비트맵 캐싱 활성화

표시 객체에 대한 비트맵 캐싱을 활성화하려면 cacheAsBitmap 속성을 true로 설정합니다.

```
mySprite.cacheAsBitmap = true;
```

cacheAsBitmap 속성을 true로 설정하면 표시 객체가 전체 좌표를 기준으로 자동으로 픽셀에 물리는 것을 볼 수 있습니다. SWF 파일을 테스트하면 복잡한 벡터 이미지에서 수행되는 애니메이션이 더욱 빠르게 렌더링되는 것이 보입니다.

다음 중 한 가지 이상의 경우에 해당되면 cacheAsBitmap이 true로 설정되었더라도 표면(캐시된 비트맵)이 만들어지지 않습니다.

- 비트맵의 높이나 폭이 2880픽셀을 초과하는 경우
- 메모리 부족으로 비트맵 할당에 실패한 경우

불투명한 배경색 설정

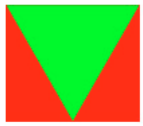
표시 객체에 불투명한 배경을 설정할 수 있습니다. 예를 들어, SWF에 복잡한 벡터 아트가 포함된 배경이 있을 경우 opaqueBackground 속성을 지정된 색상(대개 Stage와 같은 색상)으로 설정하면 됩니다. 색상은 숫자(일반적으로 16진수 색상 값)로 지정됩니다. 그러면 배경은 비트맵으로 처리되어 성능을 최적화하는 데 도움이 됩니다.

cacheAsBitmap을 true로 설정하고 또한 opaqueBackground 속성을 지정된 색상으로 설정하면, opaqueBackground 속성으로 인해 내부 비트맵이 불투명해지고 더 빠르게 렌더링됩니다. cacheAsBitmap을 true로 설정하지 않으면 opaqueBackground 속성은 표시 객체의 배경에 불투명한 벡터 사각형 모양을 추가하게 됩니다. 그러면 비트맵이 자동으로 만들어지지 않습니다.

다음 예제에서는 성능을 최적화하기 위해 표시 객체의 배경을 설정하는 방법을 보여 줍니다.

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

이 경우 myShape라는 Shape의 배경색은 빨강(0xFF0000)으로 설정됩니다. Shape 인스턴스에 녹색 삼각형 드로잉이 포함되어 있다고 가정할 때, 배경이 흰색인 Stage에서는 Shape 인스턴스의 경계 상자(모양을 완전히 감싸고 있는 사각형) 내의 공백이 빨간색인 녹색 삼각형으로 표시됩니다.



이 코드는 단색 빨강 배경을 가진 Stage에서 사용될 경우에 더 적합합니다. 다른 색상의 배경에서는 해당 색상이 대신 지정됩니다. 예를 들어, 배경이 흰색인 SWF의 경우 opaqueBackground 속성은 일반적으로 0xFFFFFF(순백색)로 설정됩니다.

블렌딩 모드 적용

블렌딩 모드에서는 하나의 이미지(기본 이미지)의 색상을 또 다른 이미지(블렌드 이미지)의 색상과 결합하여 제3의 이미지, 즉 화면에 실제로 표시되는 결과 이미지를 만들어냅니다. 기본 이미지의 개별 픽셀 값을 이에 대응되는 블렌드 이미지의 픽셀 값과 함께 처리하여 같은 위치에 새로운 픽셀 값을 산출합니다.

모든 표시 객체에는 다음 블렌딩 모드 중 하나로 설정될 수 있는 blendMode 속성이 있습니다. 이러한 값은 BlendMode 클래스에 정의된 상수입니다. 또한 상수의 실제 값인 문자열 값을 괄호로 묶어 사용할 수도 있습니다.

- BlendMode.ADD("add"): 두 이미지 사이에서 색상을 밝게 하는 디졸브 애니메이션 효과를 만드는 데 주로 사용됩니다.
- BlendMode.ALPHA("alpha"): 배경 위에 전경의 투명도를 적용하는 데 주로 사용됩니다.
- BlendMode.DARKEN("darken"): superimpose 유형에 주로 사용됩니다.
- BlendMode.DIFFERENCE ("difference"): 보다 강렬한 색상을 만드는 데 주로 사용됩니다.
- BlendMode.ERASE("erase"): 전경의 알파를 사용하여 배경의 일부를 잘라내는(지우는) 데 주로 사용됩니다.
- BlendMode.HARDLIGHT("hardlight"): 음영 효과를 만드는 데 주로 사용됩니다.
- BlendMode.INVERT("invert"): 배경을 반전시키는 데 사용됩니다.
- BlendMode.LAYER("layer"): 특정 표시 객체의 사전 합성(precomposition)을 위한 임시 버퍼를 만드는 데 사용됩니다.
- BlendMode.LIGHTEN("lighten"): superimpose 유형에 주로 사용됩니다.
- BlendMode.MULTIPLY("multiply"): 그림자 및 심도 효과를 만드는 데 주로 사용됩니다.
- BlendMode.NORMAL("normal"): 혼합 이미지의 픽셀 값이 기본 이미지의 픽셀 값을 무시하도록 지정하는 데 사용됩니다.

- BlendMode.OVERLAY("overlay"): 음영 효과를 만드는 데 주로 사용됩니다.
- BlendMode.SCREEN("screen"): 강조 및 렌즈 플레어를 만드는 데 주로 사용됩니다.
- BlendMode.SUBTRACT("subtract"): 두 이미지 사이에서 색상을 어둡게 하는 디졸브 애니메이션 효과를 만드는 데 주로 사용됩니다.

DisplayObject 색상 조절

ColorTransform 클래스의 메서드(flash.geom.ColorTransform)를 사용하여 표시 객체의 색상을 조절할 수 있습니다. 각 표시 객체에는 Transform 클래스의 인스턴스이고 표시 객체에 적용되는 다양한 변환(예: 회전, 크기 또는 위치 변경 등)에 대한 정보가 들어 있는 transform 속성이 있습니다. 기하학적 변형 외에도 Transform 클래스에는 ColorTransform 클래스의 인스턴스이고 표시 객체에 대한 색상 조절을 위한 액세스를 제공하는 colorTransform 속성이 포함되어 있습니다. 표시 객체의 색상 변형 정보에 액세스하려면 다음과 같은 코드를 사용할 수 있습니다.

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

ColorTransform 인스턴스를 만든 경우 속성 값을 보고 이미 적용된 색상 변형을 확인하거나, 해당 값을 설정하여 표시 객체의 색상을 변경할 수 있습니다. 변경 후에 표시 객체를 업데이트하려면 ColorTransform 인스턴스를 transform.colorTransform 속성에 다시 할당해야 합니다.

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// 여기에서 몇 가지 색상 변형을 만듭니다 .

// 변경 내용을 적용합니다 .
myDisplayObject.transform.colorTransform = colorInfo;
```

코드를 사용하여 색상 값 설정

ColorTransform 클래스의 color 속성을 사용하여 표시 객체에 특정 RGB(Red, Green, Blue) 색상 값을 할당할 수 있습니다. 다음 예제에서는 color 속성을 사용하여 사용자가 blueBtn 버튼을 클릭할 때 square 표시 객체의 색상을 파랑으로 변경합니다.

```
// 사각형은 스테이지에 있는 표시 객체입니다 .
// blueBtn, redBtn, greenBtn 및 blackBtn 은 스테이지에 있는 버튼입니다 .

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// 사각형과 연결된 ColorTransform 인스턴스에 액세스합니다 .
var colorInfo:ColorTransform = square.transform.colorTransform;
```

```
// blueBtn을 클릭하면 이 함수가 호출됩니다.
function makeBlue(event:MouseEvent):void
{
  // ColorTransform 객체의 색상을 설정합니다.
  colorInfo.color = 0x003399;
  // 표시 객체에 변경 내용을 적용합니다.
  square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

color 속성을 사용하여 표시 객체의 색상을 변경하면 객체에 사용된 색상 수에 관계없이 전체 객체의 색상이 완전히 변경됩니다. 예를 들어, 위에 검정 텍스트가 있는 녹색 원이 들어 있는 표시 객체가 있는 경우 해당 객체와 연관된 ColorTransform 인스턴스의 color 속성을 빨강 음영으로 설정하면 전체 객체와 원 및 텍스트가 빨강으로 변경됩니다. 따라서 텍스트가 객체의 나머지 부분과 더 이상 구분되지 않습니다.

코드를 사용하여 색상 및 밝기 효과 변경

디지털 사진처럼 여러 색상으로 된 표시 객체가 있을 때 객체의 색상을 완전히 변경하지 않고 기존 색상을 기반으로 표시 객체의 색상을 조절한다고 가정합니다. 이 경우 ColorTransform 클래스는 이러한 유형의 조정에 사용할 수 있는 일련의 승수 및 오프셋 속성을 포함합니다. redMultiplier, greenMultiplier, blueMultiplier 및 alphaMultiplier 승수 속성은 컬러 사진 필터 또는 컬러 썬글라스처럼 작동하여 표시 객체에서 특정 색상을 증가 또는 감소시킵니다. 오프셋 속성(redOffset, greenOffset, blueOffset, alphaOffset)을 사용하여 객체에 특정 색상을 추가하거나, 특정 색상이 가질 수 있는 최소값을 지정할 수 있습니다.

이러한 승수 및 오프셋 속성은 속성 관리자의 [색상] 팝업 메뉴에서 [고급]을 선택할 때 Flash 제작 도구에서 무비 클립 심볼에 사용할 수 있는 고급 색상 설정과 동일합니다.

다음 코드에서는 JPEG 이미지를 로드하고 색상 변환을 적용합니다. 이 변환에서는 마우스 포인터가 x축 및 y축을 따라 이동하면서 빨강 및 녹색 채널을 조정합니다. 이 경우 오프셋 값이 지정되어 있지 않기 때문에 화면에 표시된 각 색상 채널의 색상 값은 이미지의 원본 색상 값의 백분율입니다. 즉, 특정 픽셀에 표시되는 가장 진한 빨강 또는 녹색이 해당 픽셀의 원본 빨강 또는 녹색의 양입니다.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// 이미지를 스테이지로 로드합니다.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
```

```
this.addChild(loader);

// 로드된 이미지 위로 마우스를 이동하면 이 함수가 호출됩니다.
function adjustColor(event:MouseEvent):void
{
  // 이미지를 포함하는 Loader에 대한 ColorTransform 객체에 액세스합니다.
  var colorTransformer:ColorTransform = loader.transform.colorTransform;

  // 마우스 위치에 따라 빨강 및 녹색 승수를 설정합니다.
  // 빨강 값 범위는 0%에서 100%입니다. 커서가 왼쪽에 있을 때 0%로 빨강이 없으며,
  // 커서가 오른쪽에 있을 때 100% 빨강으로 일반적인 이미지 모양입니다.
  // 녹색 채널의 경우도 마찬가지입니다. 단, 녹색 채널은 y 축의 마우스 위치로
  // 제어된다는 점이 다릅니다.
  colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
  colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

  // 표시 객체에 변경 내용을 적용합니다.
  loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

객체 회전

rotation 속성을 사용하여 표시 객체를 회전할 수 있습니다. 이 값을 보고 객체가 회전되었는지 여부를 확인할 수 있습니다. 객체를 회전하려면 이 속성을 숫자로 설정합니다. 이 숫자는 객체에 적용할 회전 각도를 나타냅니다. 예를 들어, 다음 코드 행은 square 객체를 45도(전체 회전의 1/8) 회전합니다.

```
square.rotation = 45;
```

405페이지의 제13장, “기하 도형을 사용한 작업”에 설명된 변형 행렬을 사용하여 표시 객체를 회전할 수도 있습니다.

객체에 페이드 인/아웃 효과 적용

표시 객체의 투명도를 제어하여 객체를 부분적으로 또는 완전히 투명하게 만들 수 있습니다. 또는 투명도를 변경하여 객체에 페이드 인/아웃 효과를 나타낼 수 있습니다. DisplayObject 클래스의 alpha 속성은 표시 객체의 투명도(정확히 불투명도)를 정의합니다. alpha 속성은 0과 1 사이의 값으로 설정할 수 있습니다. 여기서 0은 완전 투명을, 1은 완전 불투명을 나타냅니다. 예를 들어, 다음 코드 행은 마우스를 클릭하면 myBall 객체를 부분적(50%)으로 투명하게 만듭니다.

```
function fadeBall(event:MouseEvent):void
{
  myBall.alpha = .5;
```

```
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

ColorTransform 클래스를 통해 사용할 수 있는 색상 조절 기능을 사용하여 표시 객체의 투명도를 변경할 수도 있습니다. 자세한 내용은 386페이지의 "DisplayObject 색상 조절"을 참조하십시오.

표시 객체 마스크 적용

표시 객체를 마스크로 사용하여 다른 표시 객체의 내용을 볼 수 있는 구멍을 만들 수 있습니다.

마스크 정의

표시 객체를 다른 표시 객체의 마스크로 지정하려면 마스크 객체를 마스크를 적용할 표시 객체의 mask 속성으로 설정합니다.

```
// 객체 maskSprite 를 객체 mySprite 의 마스크로 만듭니다 .
mySprite.mask = maskSprite;
```

마스크가 적용된 표시 객체는 마스크로 작용하는 표시 객체의 모든 불투명 영역 아래에 나타납니다. 예를 들어, 다음 코드는 100 x 100픽셀의 빨강 사각형을 포함하는 Shape 인스턴스와 반경이 25픽셀인 파랑 원을 포함하는 Sprite 인스턴스를 만듭니다. 원을 클릭하면 해당 원이 사각형에 대한 마스크로 설정되므로, 사각형에서 원으로 가려진 부분만이 표시됩니다. 즉, 빨강 원만 보이게 됩니다.

```
// 이 코드는 MovieClip 또는 Sprite 인스턴스 등의 표시 객체 컨테이너 내에서
// 실행되는 것으로 간주합니다 .

import flash.display.Shape;

// 사각형을 그려서 표시 목록에 추가합니다 .
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// 원을 그려서 표시 목록에 추가합니다 .
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
```

```
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

마스크 역할을 하는 표시 객체는 드래그할 수 있고, 애니메이션 처리하거나 동적으로 크기를 조절할 수 있으며, 단일 마스크 내에서 개별 모양을 사용할 수 있습니다. 마스크 표시 객체를 표시 목록에 추가할 필요는 없습니다. 그러나, Stage 크기를 조절할 때 마스크 객체의 크기를 조절하거나, 마스크와의 사용자 상호 작용(예: 사용자 제어 드래그 및 크기 조절)을 가능하도록 하려면 마스크 객체를 표시 목록에 추가해야 합니다. 마스크 객체를 표시 목록에 추가할 경우 표시 객체의 실제 z 인덱스(전후 순서)는 중요하지 않습니다. 마스크 객체는 화면에는 표시되지 않고 마스크로만 표시됩니다. 마스크 객체가 프레임이 여러 개 있는 MovieClip 인스턴스인 경우 마스크 객체는 마스크 역할을 하지 않을 때와 마찬가지로 해당 타임라인에 모든 프레임을 재생합니다. `mask` 속성을 `null`로 설정하여 마스크를 제거할 수 있습니다.

```
// mySprite에서 마스크를 제거합니다.
mySprite.mask = null;
```

마스크를 사용하여 다른 마스크에 마스크를 적용할 수는 없습니다. 마스크 무비 클립의 `alpha` 속성은 설정할 수 없습니다. 마스크로 사용되는 표시 객체에는 채우기만 사용되고 획은 무시됩니다.

장치 글꼴 마스크 적용

표시 객체를 사용하면 장치 글꼴로 설정된 텍스트에 마스크를 적용할 수 있습니다. 표시 객체를 사용하여 장치 글꼴에 설정된 텍스트에 마스크를 적용할 때는 마스크의 사각형 경계 상자가 마스크 모양으로 사용됩니다. 즉, 장치 글꼴 텍스트에 대해 사각형이 아닌 표시 객체 마스크를 만들면 SWF 파일에 나타나는 마스크의 모양은 마스크 자체의 모양이 아니라 마스크의 사각형 경계 상자의 모양이 됩니다.

알파 채널 마스킹

다음과 같이 알파 채널 마스킹은 마스크 표시 객체와 마스크가 적용된 표시 객체 모두 비트맵 캐싱을 사용할 경우에 지원됩니다.

```
// maskShape는 그래디언트 채우기가 적용된 Shape 인스턴스입니다.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

예를 들어, 마스크가 적용된 표시 객체에 적용되어 있는 필터와 독립적으로 마스크 객체에 필터를 사용할 경우 알파 채널 마스킹을 적용합니다.

다음 예제에서는 외부 이미지 파일을 Stage에 로드합니다. 이 이미지(정확히 말해서 이미지가 로드되는 Loader 인스턴스)는 마스크가 적용되는 표시 객체입니다. 이미지 위에 그래디언트 타원(중앙이 검정이고 가장자리로 갈수록 점점 투명해짐)이 그려집니다. 이것이 알파 마스크가 됩니다. 두 표시 객체는 모두 비트맵 캐싱 기능이 설정되어 있습니다. 타원이 이미지에 대한 마스크로 설정되고 드래그할 수 있게 됩니다.

```
// 이 코드는 MovieClip 또는 Sprite 인스턴스 등의 표시 객체 컨테이너 내에서
// 실행되는 것으로 간주합니다 .

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// 이미지를 로드하여 표시 목록에 추가합니다 .
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Sprite 를 만듭니다 .
var oval:Sprite = new Sprite();
// 그래디언트 타원을 그립니다 .
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                  colors,
                  alphas,
                  ratios,
                  matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// 표시 목록에 Sprite 를 추가합니다 .
this.addChild(oval);

// 두 표시 객체에 대해 모두 cacheAsBitmap = true 를 설정합니다 .
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// 타원을 로더와 해당 자식인 로드된 이미지의 마스크로 설정합니다 .)
loader.mask = oval;

// 타원을 드래그할 수 있게 만듭니다 .
oval.startDrag(true);
```

객체 애니메이션

애니메이션은 어떤 것을 움직이게 하거나 또는 시간에 따라 변화되도록 하는 프로세스입니다. 스크립트 애니메이션은 비디오 게임의 기본이 되는 요소이며, 기타 응용 프로그램에 세련되고 유용한 상호 작용을 추가하기 위해서도 자주 사용됩니다.

스크립트 애니메이션의 기본 아이디어는 변경이 발생해야 하며 이 변경이 시간의 경과에 따라 여러 단계로 나뉘어 진행되어야 한다는 것입니다. ActionScript에서는 일반 루프 문을 사용하여 어떤 작업을 쉽게 반복할 수 있습니다. 그러나 루프는 표시를 업데이트하기 이전에 모든 반복을 완료합니다. 스크립트 애니메이션을 만들려면 시간에 따라 일부 작업을 반복적으로 수행하고 반복이 실행될 때마다 화면을 업데이트하는 ActionScript를 작성해야 합니다.

예를 들어, 화면을 따라 공이 이동하는 간단한 애니메이션을 만든다고 가정합니다. ActionScript에는 시간 경로를 추적하여 화면을 적절하게 업데이트할 수 있는 간단한 메커니즘이 포함되어 있습니다. 즉, 공이 목표 지점에 도달할 때까지 조금씩 이동하는 코드를 작성할 수 있습니다. 이동할 때마다 화면이 업데이트되어 Stage를 가로지르는 모션이 뷰어에 표시됩니다.

실질적인 관점에서 스크립트 애니메이션을 SWF 파일의 프레임 속도(이 속도가 Flash Player에서 화면을 업데이트하는 속도이기 때문)와 동기화하여, 새 프레임이 표시될 때마다 애니메이션 변경을 하나씩 만들 수 있습니다. 각 표시 객체에는 SWF 파일의 프레임 속도에 따라 전달되는 `enterFrame` 이벤트(프레임별로 하나의 이벤트)가 있습니다. 스크립트 애니메이션을 만드는 대부분의 개발자는 `enterFrame` 이벤트를 사용하여 일정 시간 반복되는 작업을 만듭니다. `enterFrame` 이벤트를 수신하여 애니메이션 처리된 공을 프레임별로 특정 양만큼 이동하고 각 프레임에 대해 화면이 업데이트되면 공을 새 위치에 다시 그려 모션을 생성하는 코드를 작성할 수 있습니다.

참고

일정 시간 작업을 반복적으로 수행하는 다른 방법은 Timer 클래스를 사용하는 것입니다. Timer 인스턴스는 지정된 시간이 경과할 때마다 이벤트 알림을 트리거합니다. Timer 클래스의 timer 이벤트를 처리하고 시간 간격을 작게(몇 분의 일 초) 설정하여 애니메이션을 수행하는 코드를 작성할 수 있습니다. Timer 클래스 사용에 대한 자세한 내용은 186페이지의 "시간 간격 제어"를 참조하십시오.

다음 예제에서는 `circle`이라는 원 Sprite 인스턴스를 Stage에 만듭니다. 사용자가 원을 클릭하면 스크립트 애니메이션 시퀀스가 시작되어 `circle`이 완전히 투명해질 때까지 페이드됩니다(`alpha` 속성이 감소됨).

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// 원을 그려서 표시 목록에 추가합니다 .
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
```

```
circle.graphics.endFill();
addChild(circle);

// 이 애니메이션이 시작되면 프레임마다 이 함수가 호출됩니다 .
// 이 함수에 의한 변경 내용이 프레임마다 화면에 업데이트되어
// 애니메이션이 실행됩니다 .
function fadeCircle(event:Event):void
{
  circle.alpha -= .05;

  if (circle.alpha <= 0)
  {
    circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
  }
}

function startAnimation(event:MouseEvent):void
{
  circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

사용자가 원을 클릭하면 `fadeCircle()` 함수가 `enterFrame` 이벤트의 리스너로 구독되어 프레임마다 한 번씩 호출됩니다. 이 함수는 `alpha` 속성을 변경하여 `circle`에 페이드 효과를 적용하기 때문에 프레임마다 한 번씩 원의 `alpha`가 .05(5%)씩 감소하고 화면이 업데이트됩니다. `alpha` 값이 0(`circle`이 완전 투명한 상태)이 되면 `fadeCircle()` 함수가 이벤트 리스너 구독에서 제거되어 애니메이션이 종료됩니다.

예를 들어, 동일한 코드를 사용하여 페이드 효과 대신 애니메이션 모션을 만들 수 있습니다. 함수에서 `alpha`를 `enterFrame` 이벤트 리스너인 다른 속성으로 대체하면 해당 속성이 대신 애니메이션화됩니다. 예를 들어, 다음 행을

```
  circle.alpha -= .05;
```

다음 코드로 변경하면

```
  circle.x += 5;
```

x 속성이 애니메이션화되어 원이 스테이지의 오른쪽으로 이동합니다. 애니메이션이 종료되는 조건을 변경하여 원하는 x 좌표에 도달하면 애니메이션이 종료(즉, `enterFrame` 리스너 구독 해제)되도록 할 수 있습니다.

표시 내용을 동적으로 로드

다음과 같은 외부 표시 에셋을 ActionScript 3.0 응용 프로그램으로 로드할 수 있습니다.

- ActionScript 3.0에서 제작된 SWF 파일 - 이 파일은 Sprite 또는 MovieClip 클래스이거나 Sprite를 확장하는 모든 클래스가 될 수 있습니다.
- 이미지 파일 - JPG, PNG 및 GIF 파일을 포함합니다.
- AVM1 SWF 파일 - ActionScript 1.0 또는 2.0에서 작성된 SWF 파일입니다.

Loader 클래스를 사용하여 이러한 에셋을 로드합니다.

표시 객체 로드

Loader 객체는 SWF 파일과 그래픽 파일을 응용 프로그램으로 로드하는 데 사용됩니다. Loader 클래스는 DisplayObjectContainer 클래스의 하위 클래스입니다. Loader 객체는 하나의 자식 표시 객체(로드하는 SWF 또는 그래픽 파일을 나타내는 표시 객체)만 표시 목록에 포함할 수 있습니다. 다음 코드에서처럼 Loader 객체를 표시 목록에 추가할 때 로드된 자식 표시 객체를 표시 목록에 함께 추가합니다.

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

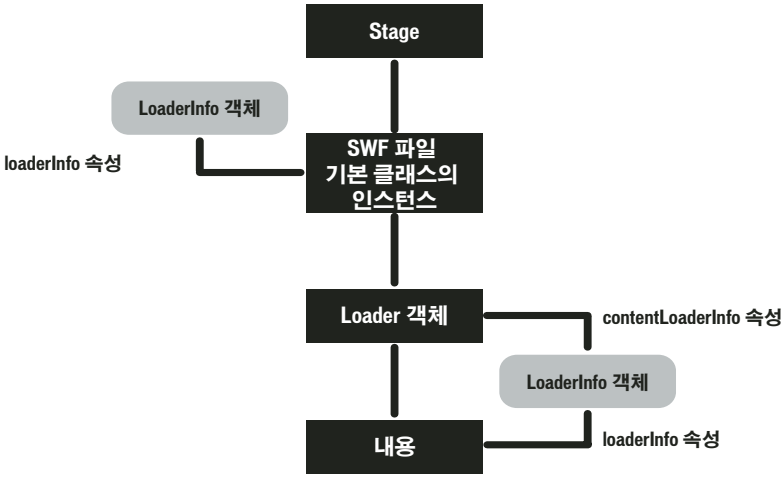
SWF 파일 또는 이미지가 로드되면 로드된 표시 객체를 다른 표시 객체 컨테이너(예: 다음 예제의 `container` DisplayObjectContainer 객체)로 이동할 수 있습니다.

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
  container.addChild(pictLdr.content);
}
```

로드 진행률 모니터링

파일 로드가 시작되면 LoaderInfo 객체가 만들어집니다. LoaderInfo 객체는 로드 진행률, 로더 및 로드 대상의 URL, 해당 미디어의 총 바이트 수, 미디어의 공칭 높이 및 폭 등과 같은 정보를 제공합니다. 또한 LoaderInfo 객체는 로드 진행률을 모니터링하는 이벤트를 전달합니다.

다음 다이어그램에서는 LoaderInfo 객체가 SWF 파일의 기본 클래스 인스턴스, Loader 객체 그리고 Loader 객체에 의해 로드된 객체에서 서로 다른 용도로 사용되는 방법을 보여 줍니다.



LoaderInfo 객체를 Loader 객체와 로드된 표시 객체 모두의 속성으로 액세스할 수 있습니다. 로드가 시작되면 Loader 객체의 `contentLoaderInfo` 속성을 통해 LoaderInfo 객체에 즉시 액세스할 수 있습니다. 표시 객체에서 로드가 완료된 경우 표시 객체의 `loaderInfo` 속성을 통해 LoaderInfo 객체를 로드된 표시 객체의 속성으로 액세스할 수도 있습니다. 로드된 표시 객체의 `loaderInfo` 속성은 Loader 객체의 `contentLoaderInfo` 속성과 동일한 LoaderInfo 객체를 나타냅니다. 즉, LoaderInfo 객체는 로드된 객체와 해당 객체를 로드한 Loader 객체 간(로더와 로드 대상 간)에 공유됩니다.

로드된 내용의 속성에 액세스하려면 다음 코드처럼 이벤트 리스너를 LoaderInfo 객체에 추가합니다.

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);
```

```
function loaded(event:Event):void
{
  var content:Sprite = event.target.content;
  content.scaleX = 2;
}
```

자세한 내용은 295페이지의 제10장, “이벤트 처리”를 참조하십시오.

로드 컨텍스트 지정

Loader 클래스의 `load()` 또는 `loadBytes()` 메서드를 통해 외부 파일을 Flash Player로 로드할 때 `context` 매개 변수를 선택적으로 지정할 수 있습니다. 이 매개 변수는 LoaderContext 객체입니다.

LoaderContext 클래스에는 로드된 내용을 사용하는 방법에 대한 컨텍스트를 정의할 수 있는 세 가지 속성이 포함되어 있습니다.

- `checkPolicyFile`: 이 속성은 SWF 파일이 아닌 이미지 파일을 로드할 때만 사용합니다. 이 속성을 `true`로 설정하면 Loader는 원래 서버에서 크로스 도메인 정책 파일을 확인합니다(714페이지의 “웹 사이트 컨트롤(크로스 도메인 정책 파일)” 참조). 이러한 설정은 Loader 객체를 포함하는 SWF 파일의 도메인과 다른 도메인에서의 내용에 대해서만 필요합니다. 서버에서 Loader 도메인에 권한을 부여하면 Loader 도메인에 있는 SWF 파일의 ActionScript는 로드된 이미지에서 데이터를 액세스할 수 있습니다.
  즉, `BitmapData.draw()` 명령을 사용하여 로드된 이미지에서 데이터를 액세스할 수 있습니다.

  Loader 객체의 도메인과 다른 도메인에서의 SWF 파일은 `Security.allowDomain()`을 호출하여 특정 도메인을 허용할 수 있습니다.
- `securityDomain`: 이 속성은 이미지가 아닌 SWF 파일을 로드할 때만 사용합니다. Loader 객체를 포함하는 파일의 도메인과 다른 도메인에서의 SWF 파일에 대해 이 속성을 지정합니다. 이 옵션을 지정하면 Flash Player에서는 크로스 도메인 정책 파일이 있는지 확인합니다. 파일이 있는 경우 크로스 정책 파일에 허용된 도메인에서의 SWF 파일은 로드된 SWF 내용을 크로스 스크립팅할 수 있습니다.
  `flash.system.SecurityDomain.currentDomain`을 이 매개 변수로 지정할 수 있습니다.
- `applicationDomain`: 이 속성은 이미지 또는 ActionScript 1.0이나 2.0으로 작성된 SWF 파일이 아닌 ActionScript 3.0으로 작성된 SWF 파일을 로드할 때만 사용합니다. 파일을 로드할 때 `applicationDomain` 매개 변수를 `flash.system.ApplicationDomain.currentDomain`으로 설정하여 Loader 객체와 동일한 응용 프로그램 도메인에 파일을 포함시키도록 지정할 수 있습니다. 로드된 SWF 파일을 동일한 응용 프로그램 도메인에 저장하면 해당 클래스에 직접 액세스할 수 있습니다. 이렇게 하면 연관된 클래스 이름을 통해 액세스할 수 있는 포함된 미디어가 들어 있는 SWF 파일을 로드할 때 유용합니다. 자세한 내용은 656페이지의 “ApplicationDomain 클래스 사용”을 참조하십시오.

다음은 다른 도메인에서 비트맵을 로드할 때 크로스 도메인 정책 파일을 확인하는 예제입니다.

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

다음은 파일을 Loader 객체와 동일한 보안 샌드박스에 저장하기 위해 다른 도메인에서 SWF를 로드할 때 크로스 도메인 정책 파일을 확인하는 예제입니다. 이 코드는 로드된 SWF 파일에 있는 클래스를 Loader 객체와 동일한 응용 프로그램 도메인에 추가합니다.

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 LoaderContext 클래스를 참조하십시오.

예제: SpriteArranger

SpriteArranger 샘플 응용 프로그램은 별도로 설명한 Geometric Shapes 샘플 응용 프로그램을 바탕으로 합니다(171페이지의 “예제: GeometricShapes” 참조).

SpriteArranger 샘플 응용 프로그램에서는 표시 객체를 처리하는 다음과 같은 다양한 개념을 설명합니다.

- 표시 객체 클래스 확장
- 표시 목록에 객체 추가
- 표시 객체 레이어 및 표시 객체 컨테이너 작업
- 표시 객체 이벤트에 응답
- 표시 객체의 속성 및 메서드 사용

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. SpriteArranger 응용 프로그램 파일은 Examples/SpriteArranger 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
SpriteArranger.mxml 또는 SpriteArranger.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/SpriteArranger/CircleSprite.as	화면에 원을 렌더링하는 Sprite 객체 유형을 정의하는 클래스입니다.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	GeometricSprite 객체를 포함하는 표시 객체 컨테이너인 canvas를 정의하는 클래스입니다.
com/example/programmingas3/SpriteArranger/SquareSprite.as	화면에 사각형을 렌더링하는 Sprite 객체 유형을 정의하는 클래스입니다.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	화면에 삼각형을 렌더링하는 Sprite 객체 유형을 정의하는 클래스입니다.
com/example/programmingas3/SpriteArranger/GeometricSprite.as	화면 상의 모양을 정의하는 데 사용되는 Sprite 객체를 확장하는 클래스로, CircleSprite, SquareSprite 및 TriangleSprite가 이 클래스를 확장합니다.
com/example/programmingas3/geometricshapes/IGeometricShape.as	모든 기하학적 모양 클래스에서 구현될 메서드를 정의하는 기본 인터페이스입니다.
com/example/programmingas3/geometricshapes/IPolygon.as	변이 여러 개인 기하학적 모양 클래스에서 구현될 메서드를 정의하는 인터페이스입니다.
com/example/programmingas3/geometricshapes/RegularPolygon.as	동일한 길이의 변이 모양의 중심을 따라 대칭적으로 배치된 기하학적 모양 유형입니다.
com/example/programmingas3/geometricshapes/Circle.as	원을 정의하는 기하학적 모양 유형입니다.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	모든 변의 길이가 같은 삼각형을 정의하는 RegularPolygon의 하위 클래스입니다.
com/example/programmingas3/geometricshapes/Square.as	네 변의 길이가 모두 같은 사각형을 정의하는 RegularPolygon의 하위 클래스입니다.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	지정된 모양 유형과 크기로 모양을 만드는 “factory 메서드”가 포함된 클래스입니다.

SpriteArranger 클래스 정의

SpriteArranger 응용 프로그램을 사용하여 화면 상의 "캔바스"에 다양한 표시 객체를 추가할 수 있습니다.

DrawingCanvas 클래스는 사용자가 화면에 모양을 추가할 수 있는 드로잉 영역, 표시 객체 컨테이너 유형 등을 정의합니다. 화면에 추가할 수 있는 모양은 GeometricSprite 클래스의 하위 클래스 중 하나의 인스턴스입니다.

DrawingCanvas 클래스

DrawingCanvas 클래스는 Sprite 클래스를 확장하며 이 상속은 다음과 같이 DrawingCanvas 클래스 선언에 정의되어 있습니다.

```
public class DrawingCanvas extends Sprite
```

Sprite 클래스는 DisplayObjectContainer 및 DisplayObject 클래스의 하위 클래스이며, DrawingCanvas 클래스는 해당 클래스의 메서드 및 속성을 사용합니다.

`DrawingCanvas()` 생성자 메서드는 Rectangle 객체인 `bounds`를 설정합니다. 이 속성은 나중에 캔바스의 윤곽을 그릴 때 사용됩니다. 그런 다음 `initCanvas()` 메서드를 다음과 같이 호출합니다.

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

다음 예제에서 보듯이 `initCanvas()` 메서드는 생성자 함수에 인수로 전달된 DrawingCanvas 객체의 다양한 속성을 정의합니다.

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

그런 다음 `initCanvas()` 메서드는 `drawBounds()` 메서드를 호출합니다. 이 메서드는 DrawingCanvas 클래스의 `graphics` 속성을 사용하여 캔바스를 그립니다. `graphics` 속성은 Shape 클래스에서 상속됩니다.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
              bounds.top - 1,
              bounds.width + 2,
              bounds.height + 2);
this.graphics.endFill();
```

응용 프로그램과의 사용자 상호 작용을 기반으로 DrawingCanvas 클래스의 다음과 같은 추가 메서드를 호출합니다.

- `addShape()` 및 `describeChildren()` 메서드(401페이지의 "캔바스에 표시 객체 추가" 참조)
- `moveToBack()`, `moveDown()`, `moveToFront()` 및 `moveUp()` 메서드(403페이지의 "표시 객체 레이어 다시 정렬" 참조)
- `onMouseUp()` 메서드(402페이지의 "표시 객체 클릭 및 드래그" 참조)

GeometricSprite 클래스 및 하위 클래스

사용자가 캔바스에 추가할 수 있는 각 표시 객체는 GeometricSprite 클래스의 다음 하위 클래스 중 하나의 인스턴스입니다.

- CircleSprite
- SquareSprite
- TriangleSprite

GeometricSprite 클래스는 flash.display.Sprite 클래스를 확장합니다.

```
public class GeometricSprite extends Sprite
```

GeometricSprite 클래스는 모든 GeometricSprite 객체에 공통되는 여러 속성을 포함합니다. 이러한 속성은 생성자 함수에서 함수에 전달된 매개 변수를 기반으로 설정됩니다. 예를 들면 다음과 같습니다.

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

GeometricSprite 클래스의 `geometricShape` 속성은 모양의 시각적 속성이 아니라 수학적 속성을 정의하는 IGeometricShape 인터페이스를 정의합니다. IGeometricShape 인터페이스를 구현하는 클래스는 GeometricShapes 샘플 응용 프로그램에 정의되어 있습니다(171페이지의 "예제: GeometricShapes" 참조).

GeometricSprite 클래스는 `drawShape()` 메서드를 정의합니다. 이 메서드는 GeometricSprite의 각 하위 클래스에서 세부적으로 재정의됩니다. 자세한 내용은 다음에 이어지는 "캔바스에 표시 객체 추가" 단원을 참조하십시오.

또한 GeometricSprite 클래스는 다음과 같은 메서드를 제공합니다.

- `onMouseDown()` 및 `onMouseUp()` 메서드(402페이지의 "표시 객체 클릭 및 드래그" 참조)
- `showSelected()` 및 `hideSelected()` 메서드(402페이지의 "표시 객체 클릭 및 드래그" 참조)

캔바스에 표시 객체 추가

사용자가 [모양 추가] 버튼을 클릭하면 응용 프로그램이 DrawingCanvas 클래스의 `addShape()` 메서드를 호출합니다. 그러면 다음 예제에서 보듯이 GeometricSprite 하위 클래스 중 하나의 해당 생성자 함수가 호출되어 새 GeometricSprite가 인스턴스화됩니다.

```
public function addShape(shapeName:String, len:Number):void
{
  var newShape:GeometricSprite;
  switch (shapeName)
  {
    case "Triangle":
      newShape = new TriangleSprite(len);
      break;

    case "Square":
      newShape = new SquareSprite(len);
      break;

    case "Circle":
      newShape = new CircleSprite(len);
      break;
  }
  newShape.alpha = 0.8;
  this.addChild(newShape);
}
```

각 생성자 메서드는 `drawShape()` 메서드를 호출합니다. 이 메서드는 Sprite 클래스에서 상속되는 클래스의 `graphics` 속성을 사용하여 해당 벡터 그래픽을 그립니다. 예를 들어, CircleSprite 클래스의 `drawShape()` 메서드는 다음과 같은 코드를 포함합니다.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

`addShape()` 함수의 끝에서 두 번째 행은 DisplayObject 클래스에서 상속되는 표시 객체의 `alpha` 속성을 설정하므로, 캔바스에 추가된 각 표시 객체는 약간 투명하여 그 뒤에 있는 객체를 볼 수 있습니다.

`addChild()` 메서드의 마지막 행은 이미 표시 목록에 있는 DrawingCanvas 클래스 인스턴스의 자식 목록에 새 표시 객체를 추가합니다. 그러면 새 표시 객체가 Stage에 표시됩니다.

응용 프로그램의 인터페이스에는 `selectedSpriteTxt`와 `outputTxt`의 두 텍스트 필드가 포함됩니다. 이러한 텍스트 필드의 텍스트 속성은 캔바스에 추가되거나 사용자가 선택한 GeometricSprite 객체에 대한 정보로 업데이트됩니다. GeometricSprite 클래스는 `toString()` 메서드를 다음과 같이 재정의하여 이 정보 보고 작업을 처리합니다.

```
public override function toString():String
```

```
{
  return this.shapeType + " of size " + this.size + " at " + this.x + ", "
  + this.y;
}
```

shapeType 속성은 각 GeometricSprite 하위 클래스의 생성자 메서드에 있는 해당 값으로 설정됩니다. 예를 들어, toString() 메서드는 DrawingCanvas 인스턴스에 최근에 추가된 CircleSprite 인스턴스에 대해 다음 값을 반환할 수 있습니다.

```
Circle of size 50 at 0, 0
```

DrawingCanvas 클래스의 describeChildren() 메서드는 DisplayObjectContainer 클래스에서 상속되는 numChildren 속성을 사용하여 for 루프에 대한 한도를 설정함으로써 캔바스의 자식 목록을 반복합니다. 다음과 같이 각 자식을 나열하는 문자열이 생성됩니다.

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
    child = this.getChildAt(i);
    desc += i + ": " + child + '\n';
}
```

결과 문자열은 outputTxt 텍스트 필드의 text 속성을 설정하는 데 사용됩니다.

표시 객체 클릭 및 드래그

사용자가 GeometricSprite 인스턴스를 클릭하면 응용 프로그램은 onMouseDown() 이벤트 핸들러를 호출합니다. 다음과 같이 이 이벤트 핸들러는 GeometricSprite 클래스의 생성자 함수에서 마우스 누름 이벤트를 수신하도록 설정됩니다.

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

그런 다음 onMouseDown() 메서드는 GeometricSprite 객체의 showSelected() 메서드를 호출합니다. 객체에 대해 이 메서드가 처음으로 호출된 경우 이 메서드는 selectionIndicator라는 새 Shape 객체를 만들고 Shape 객체의 graphics 속성을 사용하여 다음과 같이 빨간색의 강조 표시된 사각형을 그립니다.

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1,
  this.size + 1);
this.addChild(this.selectionIndicator);
```

onMouseDown() 메서드가 처음으로 호출된 경우가 아니면 이 메서드는 selectionIndicator 모양의 visible 속성(DisplayObject 클래스에서 상속됨)을 간단히 다음과 같이 설정합니다.

```
this.selectionIndicator.visible = true;
```

hideSelected() 메서드는 visible 속성을 false로 설정하여 이전에 선택된 객체의 selectionIndicator 모양을 숨깁니다.

또한 onMouseDown() 이벤트 핸들러 메서드는 Sprite 클래스에서 상속되는 startDrag() 메서드를 호출합니다. 이 메서드에는 다음과 같은 코드가 포함되어 있습니다.

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

사용자는 boundsRect 사각형에 의해 설정된 경계 내에서 선택된 객체를 캔바스 주위로 드래그할 수 있습니다.

마우스 버튼을 놓으면 mouseUp 이벤트가 전달됩니다. DrawingCanvas의 생성자 메서드는 다음과 같은 이벤트 리스너를 설정합니다.

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

이 이벤트 리스너는 개별 GeometricSprite 객체가 아니라 DrawingCanvas 객체에 대해 설정됩니다. GeometricSprite 객체가 드래그될 때 마우스 버튼을 놓으면 다른 표시 객체(다른 GeometricSprite 객체) 뒤에서 드래그가 중단될 수 있기 때문입니다. 전경의 표시 객체는 마우스 놓음 이벤트를 수신하지만 사용자가 드래그 중인 표시 객체는 이 이벤트를 수신하지 못합니다. DrawingCanvas 객체에 리스너를 추가하면 이벤트가 항상 처리되도록 할 수 있습니다.

onMouseUp() 메서드는 GeometricSprite 객체의 onMouseUp() 메서드를 호출하고, 이 메서드는 GeometricSprite 객체의 stopDrag() 메서드를 호출하게 됩니다.

표시 객체 레이어 다시 정렬

응용 프로그램의 사용자 인터페이스에는 Move Back, Move Down, Move Up, Move to Front라는 버튼이 있습니다. 사용자가 이 버튼 중 하나를 클릭하면 응용 프로그램은 DrawingCanvas 클래스의 해당 메서드(moveToBack(), moveDown(), moveUp() 또는 moveToFront())를 호출합니다. 예를 들어, moveToBack() 메서드는 다음과 같은 코드를 포함합니다.

```
public function moveToBack(shape:GeometricSprite):void
{
  var index:int = this.getChildIndex(shape);
  if (index > 0)
  {
    this.setChildIndex(shape, 0);
  }
}
```

이 메서드는 DisplayObjectContainer 클래스에서 상속되는 setChildIndex() 메서드를 사용하여 DrawingCanvas 인스턴스(this)의 자식 목록에서 인덱스 위치 0에 표시 객체를 배치합니다.

moveDown() 메서드는 DrawingCanvas 인스턴스의 자식 목록에서 표시 객체의 인덱스 위치가 1씩 감소된다는 점을 제외하고는 유사하게 작동합니다.

```
public function moveDown(shape:GeometricSprite):void
{
```

```
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
      this.setChildIndex(shape, index - 1);
    }
}
```

moveUp() 및 moveToFront() 메서드는 moveToBack() 및 moveDown() 메서드와 유사하게 작동합니다.

제 13 장

13 기하 도형을 사용한 작업

flash.geom 패키지에는 점, 사각형, 변형 행렬 등의 기하학적 객체를 정의하는 클래스가 포함되어 있습니다. 이러한 클래스를 사용하여 다른 클래스에 사용되는 객체의 속성을 정의할 수 있습니다.

목차

기하 도형의 기초

기하 도형을 사용한 작업 소개

기하학은 많은 사람들에게 학창 시절에 최소한으로 수강하고 지나치고 싶어했던 과목일지는 모르지만, ActionScript에서는 약간의 기하학 지식이 많은 도움이 됩니다.

flash.geom 패키지에는 점, 사각형, 변형 행렬 등의 기하학적 객체를 정의하는 클래스가 포함되어 있습니다. 이러한 클래스는 자체적인 기능이 있는 것은 아니지만 다른 클래스에서 사용되는 객체 속성을 정의하는 데 사용됩니다.

모든 기하 도형 클래스는 화면에 표시되는 위치가 2차원 평면으로 나타난다는 개념을 기본으로 합니다. 화면을 가로축(x)과 세로축(y)이 있는 평면 그래프로 생각할 수 있습니다. 화면의 모든 위치(또는 *점*)는 x, y 값 쌍, 즉 위치 *좌표*로 나타낼 수 있습니다.

Stage를 비롯한 모든 표시 객체에는 각각 *좌표 공간*이 지정되어 있으며 이 좌표 공간을 사용하여 자식 표시 객체, 드로잉 등의 위치를 계획하게 됩니다. 일반적으로 표시 객체의 좌측 상단 모서리가 *원점*(x축 및 y축이 만나는 0, 0 좌표)으로 설정됩니다. Stage에서는 이 사실이 항상 적용되지만 다른 표시 객체의 경우에는 그렇지 않을 수도 있습니다. 표준 2차원 좌표계에서처럼 x축의 값이 커질수록 위치가 오른쪽으로 이동하고 x축의 값이 작아질수록 왼쪽으로 이동합니다. 마찬가지로 원점을 기준으로 왼쪽에 위치하는 점의 x 좌표는 음의 값을 갖습니다. 그러나 일반적인 좌표계와는 반대로 y축 ActionScript 값이 커질수록 위치가 화면 아래쪽으로 내려가고 작아질수록 화면 위쪽으로 올라가며, 원점을 기준으로 위쪽에 위치하는 점의 y 좌표가 음의 값을 갖습니다. 스테이지의 왼쪽 위 모서리가 해당 좌표 공간의 원점이기 때문에 스테이지의 모든 객체는 0보다 크고 스테이지 너비보다 작은 x 좌표값과 0보다 크고 스테이지 높이보다 작은 y 좌표값을 가집니다.

Point 클래스 인스턴스를 사용하여 좌표 공간에 있는 각각의 점을 나타낼 수 있습니다. 또한 Rectangle 인스턴스를 만들어 좌표 공간을 구성하는 사각형 영역을 나타낼 수 있습니다. 고급 사용자의 경우, Matrix 인스턴스를 사용하여 표시 객체에 여러 가지 복잡한 변형을 적용할 수도 있습니다. 회전, 위치, 배율 변경 등 상당수의 간단한 변형은 객체 속성을 사용하여 표시 객체에 직접 적용할 수 있습니다. 표시 객체 속성을 사용한 변형 적용에 대한 자세한 내용은 372페이지의 “표시 객체 조작”을 참조하십시오.

일반적인 기하 도형 작업

ActionScript에서 기하 도형 클래스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- 두 점 사이의 거리 계산
- 서로 다른 좌표 공간에 있는 점의 좌표 구하기
- 각도 및 거리를 사용하여 표시 객체 이동
- Rectangle 인스턴스 작업:
  - Rectangle 인스턴스 위치 변경
  - Rectangle 인스턴스 크기 변경
  - Rectangle 인스턴스의 전체 크기 또는 중복 영역 구하기
- Matrix 객체 만들기
- Matrix 객체를 사용하여 표시 객체에 변형 적용

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 직교 좌표: 좌표가 일반적으로 숫자 쌍(예: 5, 12 또는 17, -23)으로 나타납니다. 두 숫자는 각각 x 좌표와 y 좌표를 나타냅니다.
- 좌표 공간: 표시 객체에 포함된 좌표의 그래프이며 해당 객체의 자식 요소가 배치됩니다.
- 원점: 좌표 공간에서 x축과 y축이 만나는 점을 나타냅니다. 이 점의 좌표는 0, 0으로 표시됩니다.
- 점: 좌표 공간에서의 한 위치를 나타냅니다. ActionScript에 사용되는 2차원 좌표계에서는 x축과 y축(점의 좌표) 위의 해당 위치로 점이 정의됩니다.
- 등록 포인트: 표시 객체에서 좌표 공간의 원점(0, 0 좌표)입니다.
- 배율: 객체의 원래 크기에 비례한 크기를 나타냅니다. 동사로 '크기 조정'이라고도 하는데, 이 경우에는 객체를 늘리거나 줄여 크기를 조절한다는 의미를 갖습니다.
- 평행 이동: 한 좌표 공간에서 다른 좌표 공간으로 점의 좌표를 변경합니다.
- 변형: 객체 회전, 크기 변경, 모양 기울이기 또는 왜곡, 색상 변경 등 그래픽의 시각적 특징을 조절합니다.
- X축: ActionScript에서 사용하는 2D 좌표계의 수평 축입니다.
- Y축: ActionScript에서 사용하는 2D 좌표계의 수직 축입니다.

이 장의 예제를 사용하여 작업

이 장의 예제에서는 계산 또는 값 변경을 보여 주고, 대부분의 예제에 해당 `trace()` 함수 호출이 포함되어 코드 결과를 보여 줍니다. 이 예제를 테스트하려면 다음을 수행합니다.

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널에서 코드 샘플의 `trace()` 함수에 대한 결과를 확인합니다.

이 장의 일부 예제에서는 표시 객체에 변형 적용 방법을 보여 줍니다. 이러한 예제의 결과는 텍스트 출력을 이용하지 않고 시각적으로 확인할 수 있습니다. 변형 예제를 테스트하려면 다음을 수행합니다.

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. 스테이지에 무비 클립 심볼 인스턴스를 만듭니다. 예를 들어 모양을 하나 그려 그 모양을 선택한 다음 [수정] > [심볼로 변환]을 선택한 후, 심볼에 이름을 지정합니다.

5. 스테이지 무비 클립을 선택한 상태에서 속성 관리자에서 무비 클립에 인스턴스 이름을 지정합니다. 이 이름은 예제 코드 샘플에서 해당 표시 객체에 대해 사용한 이름과 이름과 일치해야 합니다. 예를 들어, 코드 샘플에서 `myDisplayObject`라는 객체를 조작하려는 경우 해당 무비 클립 인스턴스 이름도 `myDisplayObject`로 지정해야 합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   코드 샘플에 지정된 대로 객체에 적용한 변형 결과가 화면에 표시됩니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

Point 객체 사용

Point 객체는 직교 좌표 쌍을 정의합니다. 이 객체는 2차원 좌표계에서의 위치를 나타냅니다. 여기에서 *x*는 가로 축을 나타내고 *y*는 세로 축을 나타냅니다.

Point 객체를 정의하려면 다음과 같이 `x` 및 `y` 속성을 설정해야 합니다.

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

두 점 사이의 거리 확인

Point 클래스의 `distance()` 메서드를 사용하면 좌표 공간에서 두 점 사이의 거리를 확인할 수 있습니다. 예를 들어, 다음 코드에서는 같은 표시 객체 컨테이너에 있는 두 표시 객체(`circle1`과 `circle2`)의 등록 포인트 사이의 거리를 확인합니다.

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

좌표 공간 평행 이동

두 표시 객체가 서로 다른 표시 객체 컨테이너에 있는 경우에는 객체가 서로 다른 좌표 공간에 있을 수 있습니다. 이런 경우 DisplayObject 클래스의 `localToGlobal()` 메서드를 사용하여 좌표를 Stage의 전역 좌표 공간으로 평행 이동할 수 있습니다. 예를 들어, 다음 코드에서는 서로 다른 표시 객체 컨테이너에 있는 두 표시 객체(`circle1`과 `circle2`)의 등록 포인트 사이의 거리를 확인합니다.

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
```

```
var pt2:Point = new Point(circle1.x, circle1.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

마찬가지로 Stage의 특정 지점에서 `target`이라는 표시 객체의 등록 포인트 거리를 확인하려면 DisplayObject 클래스의 `localToGlobal()` 메서드를 사용하면 됩니다.

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

각도 및 거리를 지정하여 표시 객체 이동

Point 클래스의 `polar()` 메서드를 사용하면 표시 객체를 특정 각도 및 거리만큼 이동할 수 있습니다. 예를 들어, 다음 코드에서는 `myDisplayObject` 객체를 60도 방향으로 100픽셀만큼 이동합니다.

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Point 클래스의 다른 사용 방법

다음과 같은 메서드 및 속성과 함께 Point 객체를 사용할 수 있습니다.

클래스	메서드 또는 속성	설명
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()` `getObjectsUnderPoint()`	표시 객체 컨테이너 내의 지점에서 객체 목록을 반환하는 데 사용됩니다.
BitmapData	`hitTest()`	BitmapData 객체 내의 픽셀 및 히트 검사 대상 지점을 정의하는 데 사용됩니다.

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		작업을 정의하는 사각형 위치를 정의하는 데 사용됩니다.

클래스	메서드 또는 속성	설명
Matrix	`deltaTransformPoint()` `transformPoint()`	변환을 적용할 지점을 정의하는 데 사용됩니다.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		이러한 속성을 정의하는 데 사용됩니다.

Rectangle 객체 사용

Rectangle 객체는 사각형 영역을 정의합니다. 이 객체에는 왼쪽 위 모서리의 *x* 및 *y* 좌표로 정의된 위치, `width` 속성, `height` 속성 등이 있습니다. 다음과 같이 `Rectangle()` 생성자 함수를 호출하여 새 Rectangle 객체에 대해 이러한 속성을 정의할 수 있습니다.

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Rectangle 객체의 크기 및 위치 조정

다양한 방법을 사용하여 Rectangle 객체의 크기와 위치를 조정할 수 있습니다.

`x` 및 `y` 속성을 변경하면 Rectangle 객체의 위치를 직접 조정할 수 있습니다. 이때 Rectangle 객체의 폭이나 높이에는 아무런 영향을 주지 않습니다.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

다음 코드와 같이 Rectangle 객체의 `left` 또는 `top` 속성을 변경하면 객체의 위치가 조정되고 각각 `left` 및 `top` 속성과 일치하는 `x` 및 `y` 속성이 변경됩니다. 그러나 Rectangle 객체의 왼쪽 아래 모서리 위치는 변경되지 않고 크기만 조정됩니다.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
```

```
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=30, y=20, w=70, h=30)
```

또한 다음 예제와 같이 Rectangle 객체의 `bottom` 또는 `right` 속성을 변경하면 왼쪽 위 모서리 위치는 변경되지 않고 크기만 적절하게 조정됩니다.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

다음과 같이 `offset()` 메서드를 사용하여 Rectangle 객체의 위치를 조정할 수도 있습니다.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

`offsetPt()` 메서드도 비슷한 방식으로 작동하지만 *x* 및 *y* 오프셋 값 대신 Point 객체를 매개 변수로 사용한다는 차이가 있습니다.

`dx`와 `dy`라는 두 개의 매개 변수를 포함하는 `inflate()` 메서드를 사용하여 Rectangle 객체의 크기를 조정할 수도 있습니다. `dx` 매개 변수는 사각형의 왼쪽 및 오른쪽 변이 중심에서 이동하는 픽셀 크기를 나타내고, `dy` 매개 변수는 사각형의 위쪽 및 아래쪽 변이 중심에서 이동하는 픽셀 크기를 나타냅니다.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

`inflatePt()` 메서드도 비슷한 방식으로 작동하지만 `dx` 및 `dy` 값 대신 Point 객체를 매개 변수로 사용한다는 차이가 있습니다.

Rectangle 객체의 통합 및 교차 영역 얻기

union() 메서드를 사용하면 두 사각형의 경계선에 의해 형성되는 사각형 영역을 얻을 수 있습니다.

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

intersection() 메서드를 사용하면 두 사각형의 서로 겹치는 영역에 의해 형성되는 사각형 영역을 얻을 수 있습니다.

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

두 사각형이 교차하는지 여부를 확인하려면 intersects() 메서드를 사용합니다. 또한 표시 객체가 Stage의 특정 영역 내에 있는지 여부를 확인하는 경우에도 intersects() 메서드를 사용할 수 있습니다. 예를 들어, 다음 코드에서는 circle 객체를 포함하는 표시 객체 컨테이너의 좌표 공간이 Stage의 좌표 공간과 같다고 가정합니다. 이 예제에서는 intersects() 메서드를 사용하여 표시 객체 circle이 target1과 target2라는 Rectangle 객체로 정의된 스테이지 특정 영역과 교차하는지 여부를 확인하는 방법을 보여 줍니다.

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

마찬가지로 두 표시 객체의 경계 사각형이 교차하는지 여부를 확인하는 경우에도 intersects() 메서드를 사용할 수 있습니다. DisplayObject 클래스의 getRect() 메서드를 사용하면 표시 객체의 획으로 인해 경계 영역에 추가되는 공간을 포함할 수 있습니다.

Rectangle 객체의 다른 사용 방법

다음과 같은 메서드 및 속성과 함께 Rectangle 객체를 사용할 수 있습니다.

클래스	메서드 또는 속성	설명
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()`, `threshold()`	BitmapData 객체의 영역을 정의하기 위해 일부 매개 변수의 유형으로 사용됩니다.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	속성의 데이터 유형 또는 반환되는 데이터 유형으로 사용됩니다.
PrintJob	`addPage()`	`printArea` 매개 변수를 정의하는 데 사용됩니다.
Sprite	`startDrag()`	`bounds` 매개 변수를 정의하는 데 사용됩니다.
TextField	`getCharBoundaries()`	반환값 유형으로 사용됩니다.
Transform	`pixelBounds`	데이터 유형으로 사용됩니다.

Matrix 객체 사용

Matrix 클래스는 특정 좌표 공간의 점을 다른 좌표 공간에 매핑하는 방법을 결정하는 변형 행렬을 나타냅니다. Matrix 객체의 속성을 설정하여 Matrix 객체를 Transform 객체의 `matrix` 속성에 적용한 다음, 이 Transform 객체를 표시 객체의 `transform` 속성으로 적용하면 표시 객체에서 다양한 그래픽 변환 작업을 수행할 수 있습니다. 이러한 변환 기능에는 평행 이동(*x* 와 *y* 위치 변경), 회전, 크기 조절, 기울이기 등이 있습니다.

Matrix 객체 정의

Matrix 객체의 속성(`a`, `b`, `c`, `d`, `tx`, `ty`)을 직접 조정하여 행렬을 정의할 수도 있지만 `createBox()` 메서드를 사용하면 더 쉽습니다. 이 메서드에는 결과 행렬의 크기 조절, 회전 및 평행 이동 효과를 직접 정의할 수 있는 매개 변수가 포함되어 있습니다. 예를 들어, 다음 코드를 사용하면 가로 방향으로 2.0배 크기 조절, 세로 방향으로 3.0배 크기 조절, 45도 각도로 회전, 오른쪽으로 10픽셀 이동, 아래쪽으로 20픽셀 이동하는 효과가 적용된 Matrix 객체가 만들어집니다.

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

`scale()`, `rotate()` 및 `translate()` 메서드를 사용하여 Matrix 객체의 크기 조절, 회전 및 평행 이동 효과를 조정할 수도 있습니다. 이러한 메서드는 기존 Matrix 객체의 값을 결합합니다. 예를 들어 다음 코드에서는 `scale()` 메서드와 `rotate()` 메서드가 두 번 호출되므로 Matrix 객체가 4배로 확대되고 60도 회전합니다.

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Matrix 객체에 기울이기 변환을 적용하려면 b 또는 c 속성을 조정해야 합니다. b 속성을 조정하면 행렬이 세로 방향으로 기울어지고 c 속성을 조정하면 행렬이 가로 방향으로 기울어집니다. 다음 코드에서는 myMatrix라는 Matrix 객체를 세로 방향, 2의 배율로 기울입니다.

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

표시 객체의 transform 속성에 Matrix 변환을 적용할 수 있습니다. 예를 들어 다음 코드에서는 myDisplayObject라는 표시 객체에 행렬 변환을 적용합니다.

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

첫 번째 행에서는 Matrix 객체를 myDisplayObject 표시 객체에서 사용하는 기존 변형 행렬로 설정합니다(myDisplayObject 표시 객체에 대한 transformation 속성의 matrix 속성). 따라서 사용자가 호출하는 Matrix 클래스 메서드에는 표시 객체의 기존 위치, 크기 조절 및 회전 효과가 누적 적용됩니다.

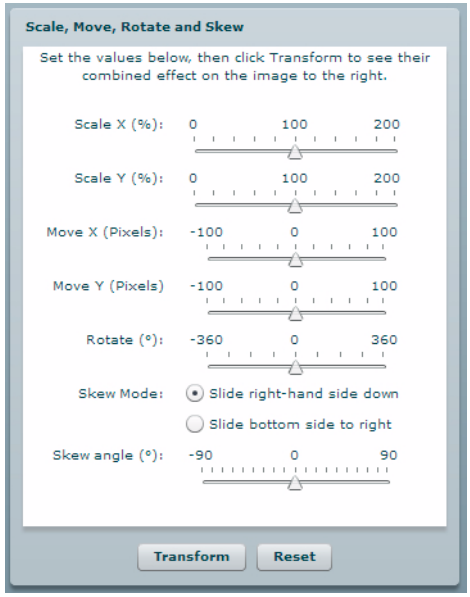
참고 flash.geometry 패키지에는 ColorTransform 클래스도 포함되어 있습니다. 이 클래스는 Transform 객체의 colorTransform 속성을 설정하는 데 사용되지만 기하학적 변환을 적용하지 않으므로 이 장에서는 설명하지 않습니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 ColorTransform 클래스를 참조하십시오.

예제: 표시 객체에 행렬 변환 적용

DisplayObjectTransformer 샘플 응용 프로그램에서는 다음과 같이 Matrix 클래스를 사용하여 표시 객체를 변환하는 다양한 기능을 보여 줍니다.

- 표시 객체 회전
- 표시 객체 크기 조절
- 표시 객체 평행 이동(위치 변경)
- 표시 객체 기울이기

이 응용 프로그램에는 다음과 같이 행렬 변환 매개 변수를 조정할 수 있는 인터페이스가 있습니다.



사용자가 [Transform] 버튼을 클릭하면 응용 프로그램에서 적절한 변환을 적용합니다.

원본 표시 객체와 -45° 회전 및 50% 축소 효과가 적용된 표시 객체

이 샘플에 대한 응용 프로그램 파일을 구하려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 방문하십시오. DisplayObjectTransformer 응용 프로그램 파일은 Samples/DisplayObjectTransformer 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
DisplayObjectTransformer.mxml 또는 DisplayObjectTransformer.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/geometry/MatrixTransformer.as	행렬 변환을 적용하는 데 필요한 메서드가 포함된 클래스입니다.
img/	응용 프로그램에 사용되는 샘플 이미지 파일이 포함된 디렉토리입니다.

MatrixTransformer 클래스 정의

MatrixTransformer 클래스에는 Matrix 객체의 기하학적 변환을 적용하는 정적 메서드가 포함되어 있습니다.

transform() 메서드

`transform()` 메서드에는 다음 항목에 대한 각 매개 변수가 포함되어 있습니다.

- `sourceMatrix`—이 메서드로 변환하는 입력 행렬
- `xScale` 및 `yScale`—*x* 및 *y* 배율 인수
- `dx` 및 `dy`—*x* 및 *y* 평행 이동 크기(픽셀)
- `rotation`—회전 크기(도)
- `skew`—기울이기 인수(백분율)
- `skewType`—기울이기 방향(`"right"` 또는 `"left"`)

반환값은 결과 행렬입니다.

`transform()` 메서드는 클래스의 다음 정적 메서드를 호출합니다.

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

각 메서드는 변환 효과가 적용된 소스 행렬을 반환합니다.

skew() 메서드

skew() 메서드는 행렬의 b 및 c 속성을 조정하여 행렬을 기울입니다. 선택적 매개 변수인 unit은 기울이기 각도를 정의하는 데 사용되는 단위를 결정하고, 필요한 경우 이 메서드는 angle 값을 라디안으로 변환합니다.

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
```

기울이기 변환을 적용하기 위해 skewMatrix라는 Matrix 객체를 만들어 조정합니다. 처음에는 다음과 같이 단위 행렬입니다.

```
var skewMatrix:Matrix = new Matrix();
```

skewSide 매개 변수는 기울이기 변환을 적용할 변을 결정합니다. 이 매개 변수가 "right"로 설정되면 다음 코드에서는 행렬의 b 속성을 설정합니다.

```
skewMatrix.b = Math.tan(angle);
```

그렇지 않은 경우에는 다음과 같이 Matrix 객체의 c 속성을 조정하여 아래쪽 변이 기울어집니다.

```
skewMatrix.c = Math.tan(angle);
```

그런 다음 아래 예제와 같이 두 행렬을 연결하여 기존 행렬에 결과 기울이기가 적용됩니다.

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

scale() 메서드

다음 예제와 같이 scale() 메서드는 먼저 배율 인수가 백분율로 지정되어 있는 경우 이를 조정하고 행렬 객체의 scale() 메서드를 사용합니다.

```
if (percent)
{
  xScale = xScale / 100;
  yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

translate() 메서드

translate() 메서드는 다음과 같이 행렬 객체의 translate() 메서드를 호출하여 dx 및 dy 평행 이동 인수를 적용합니다.

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

rotate() 메서드

rotate() 메서드는 입력 회전 인수가 도 또는 그래디언트 단위로 지정되어 있는 경우 이를 라디안으로 변환한 다음, 행렬 객체의 rotate() 메서드를 호출합니다.

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

응용 프로그램에서 MatrixTransformer.transform() 메서드 호출

응용 프로그램에는 사용자가 변환 매개 변수를 입력할 수 있는 사용자 인터페이스가 포함되어 있습니다. 매개 변수가 입력되면 응용 프로그램에서는 다음과 같이 표시 객체에 대한 transform 속성의 matrix 속성과 함께 해당 매개 변수를 Matrix.transform() 메서드에 전달합니다.

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
                                         xScaleSlider.value,
                                         yScaleSlider.value,
                                         dxSlider.value,
                                         dySlider.value,
                                         rotationSlider.value,
                                         skewSlider.value,
                                         skewSide );
```

그런 다음 표시 객체에 대한 transform 속성의 matrix 속성에 반환값을 적용하여 변환을 트리거합니다.

```
img.content.transform.matrix = tempMatrix;
```

제 14 장

14 드로잉 API 사용

가져온 이미지 및 아트웍도 중요하지만 드로잉 API 기능을 사용하면 ActionScript에서 선과 도형을 그릴 수 있으므로 컴퓨터에서 빈 캔바스처럼 응용 프로그램을 시작하여 원하는 이미지를 만들 수 있습니다. 그래픽을 직접 작성할 수 있는 기능은 응용 프로그램의 무한한 가능성을 열어줍니다. 이 장에서 다루는 기술을 통해 드로잉 프로그램을 생성하거나 움직이는 대화형 아트를 만들고 사용자 인터페이스 요소를 프로그래밍 방식으로 작성할 수 있습니다.

목차

드로잉 API 사용의 기초

드로잉 API 사용 소개

드로잉 API는 ActionScript를 사용하여 벡터 그래픽(선, 곡선, 모양, 채우기, 그래디언트)을 만들고 화면에 이를 표시하는 ActionScript 내장 기능의 이름입니다. flash.display.Graphics 클래스가 이 기능을 제공합니다. 이러한 각 클래스에 정의된 `graphics` 속성을 사용하여 Shape, Sprite 또는 MovieClip 인스턴스에서 ActionScript로 그림을 그릴 수 있습니다. 이러한 각 클래스의 `graphics` 속성은 실제로 Graphics 클래스의 인스턴스입니다.

코드를 사용한 드로잉에 이제 막 입문한 사용자를 위해 Graphics 클래스에는 원, 타원, 사각형, 모서리가 둥근 사각형과 같은 일반 도형을 쉽게 그릴 수 있는 몇 가지 메서드가 포함되어 있습니다. 이러한 도형은 빈 선 또는 채워진 모양으로 그릴 수 있습니다. 고급 기능을 필요로 하는 사용자를 위해 Graphics 클래스에는 선과 이차 베지어 곡선을 그릴 수 있는 메서드도 포함되어 있습니다. 이러한 메서드는 Math 클래스의 삼각 함수와 함께 사용하여 필요한 도형을 만들 수 있습니다.

일반적인 드로잉 API 작업

다음은 ActionScript에서 드로잉 API를 사용하여 수행할 수 있는 작업들로, 이 장에서 설명하는 내용이기도 합니다.

- 모양 그리기를 위한 선 스타일 및 채우기 스타일 정의
- 직선 및 곡선 그리기
- 원, 타원, 사각형 등의 모양을 그리기 위한 메서드 사용
- 그래디언트 선 및 채우기를 사용하여 그리기
- 그래디언트 생성을 위한 행렬 정의
- 드로잉 API에서 삼각 함수 사용
- 드로잉 API를 애니메이션으로 통합

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 앵커 포인트: 이차 베지어 곡선 두 끝점 중 하나입니다.
- 제어 포인트: 이차 베지어 곡선의 방향 및 굴곡 정도를 정의하는 포인트입니다. 곡선이 제어 포인트에 닿지는 않지만 제어 포인트를 향하는 곡선이 그려집니다.
- 좌표 공간: 표시 객체에 포함된 좌표의 그래프이며 해당 객체의 자식 요소가 배치됩니다.
- 채우기: 색상선으로 그린 모양에서 단색의 내부 또는 외곽선이 없는 모양 전체를 말합니다.

- 그래디언트: 한 가지 색상에서 하나 이상의 다른 색상으로 점진적으로 변하는 색상입니다(단색의 반대).
- 점: 좌표 공간에서의 한 위치를 나타냅니다. ActionScript에 사용되는 2차원 좌표계에서는 x축과 y축(점의 좌표) 위의 해당 위치로 점이 정의됩니다.
- 이차 베지어 곡선: 특정 수학 공식으로 정의된 곡선 유형입니다. 이 유형의 곡선에서 곡선의 모양은 앵커 포인트의 위치(곡선의 두 끝점)와 곡선의 방향 및 굴곡 정도를 정의하는 제어 포인트를 기준으로 계산됩니다.
- 배율: 객체의 원래 크기에 비례한 크기를 나타냅니다. 동사로 '크기 조정'이라고도 하는데, 이 경우에는 객체를 늘리거나 줄여 크기를 조절한다는 의미를 갖습니다.
- 획: 색상선으로 그린 모양에서 외곽선 부분 또는 채워지지 않은 모양의 선입니다.
- 평행 이동: 한 좌표 공간에서 다른 좌표 공간으로 점의 좌표를 변경합니다.
- X축: ActionScript에서 사용하는 2D 좌표계의 수평 축입니다.
- Y축: ActionScript에서 사용하는 2D 좌표계의 수직 축입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장은 시각적 내용 그리기를 다루므로, 코드 샘플을 테스트하려면 코드를 실행한 후 생성된 SWF의 결과를 확인해야 합니다. 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   생성된 SWF 파일에서 코드 샘플의 결과를 확인합니다.

Graphics 클래스 이해

각 Shape, Sprite 및 MovieClip 객체에는 Graphics 클래스의 인스턴스인 `graphics` 속성이 있습니다. Graphics 클래스에는 선, 채우기 및 모양을 그리기 위한 메서드와 속성이 포함됩니다. 내용을 그리기 위한 캔바스로만 표시 객체를 사용하려는 경우에는 Shape 인스턴스를 사용할 수 있습니다. Shape 인스턴스는 Sprite 및 MovieClip 클래스의 추가 기능에 대한 오버헤드가 없으므로 다른 그리기용 표시 객체보다 성능이 뛰어납니다. 그래픽 내용을 그릴 수 있는 표시 객체가 필요하고 이 표시 객체가 다른 표시 객체를 포함하도록 하려는 경우에는 Sprite 인스턴스를 사용할 수 있습니다. 다양한 작업에 사용할 표시 객체를 결정하는 자세한 방법은 370페이지의 "DisplayObject 하위 클래스 선택"을 참조하십시오.

선 및 곡선 그리기

Graphics 인스턴스를 사용하여 수행되는 모든 드로잉은 선 및 곡선이 있는 기본 드로잉을 기반으로 합니다. 따라서 모든 ActionScript 드로잉은 다음의 동일한 단계들을 통해 수행되어야 합니다.

- 선 및 채우기 스타일 정의
- 초기 드로잉 위치 설정
- 선, 곡선 및 모양 그리기(선택적으로 드로잉 포인트 이동)
- 채우기 생성 완료(필요한 경우)

선 및 채우기 스타일 정의

Shape, Sprite 또는 MovieClip 인스턴스의 `graphics` 속성을 사용하여 그리려면 드로잉에 사용할 스타일(선 크기 및 색상, 채우기 색상)을 먼저 정의해야 합니다. Adobe Flash CS3 Professional 또는 다른 드로잉 응용 프로그램의 드로잉 도구를 사용할 때와 마찬가지로, ActionScript를 사용하여 그릴 때는 획이나 채우기 색상을 사용할 수도 있고 사용하지 않을 수도 있습니다. 획 모양은 `lineStyle()` 또는 `lineGradientStyle()` 메서드를 사용하여 지정하고, 실선을 만들려면 `lineStyle()` 메서드를 사용합니다. 이 메서드를 호출할 때 가장 일반적으로 지정하는 값은 처음 3개의 매개 변수인 선 두께, 색상 및 알파입니다. 예를 들어, 다음 코드 행은 `myShape`라는 Shape가 2픽셀 두께, 빨강(0x990000) 및 불투명도 75%의 선을 그리도록 지시합니다.

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

알파 매개 변수의 기본값은 1.0(100%)이며, 완전히 불투명한 선을 원하는 경우 이 매개 변수를 해제할 수 있습니다. `lineStyle()` 메서드는 또한 픽셀 힌트 및 크기 조절 모드를 위한 두 개의 추가 매개 변수를 사용합니다. 이러한 매개 변수 사용에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 `Graphics.lineStyle()` 메서드에 대한 설명을 참조하십시오.

그래디언트 선을 만들려면 `lineGradientStyle()` 메서드를 사용합니다. 이 방법에 대해서는 428페이지의 "그래디언트 선 및 채우기 만들기"에 설명되어 있습니다.

채워진 모양을 만들려면 드로잉을 시작하기 전에 `beginFill()`, `beginGradientFill()` 또는 `beginBitmapFill()` 메서드를 호출하십시오. 그 중에서 가장 기본적인 `beginFill()` 메서드는 채우기 색상과 채우기 색상에 대한 알파 값(옵션)이라는 두 개의 매개 변수를 사용합니다. 예를 들어, 녹색으로 채워진 모양을 그리는 경우 다음과 같은 코드를 사용할 수 있습니다(`myShape`라는 객체에서 그린다고 가정).

```
myShape.graphics.beginFill(0x00FF00);
```

채우기 메서드를 호출하면 이전의 모든 채우기가 암시적으로 종료된 후 새 메서드가 시작됩니다. 획 스타일을 지정하는 메서드를 호출하면 이전 획이 대체되지만, 이전에 지정한 채우기는 바뀌지 않으며 그 반대의 경우도 마찬가지입니다.

선 스타일 및 채우기 속성을 지정했으면 다음 단계는 드로잉 시작점을 지정하는 것입니다. Graphics 인스턴스에는 종이 위의 펜촉과 같은 드로잉 포인트가 있습니다. 드로잉 포인트가 있는 위치가 다음 드로잉 액션이 시작되는 지점이 됩니다. 처음에 Graphics 객체는 드로잉 포인트 0에서 시작하는데, 0은 드로잉이 수행되는 객체의 좌표 공간에서의 시작점입니다. 다른 포인트에서 그리기를 시작하려면 `moveTo()` 메서드를 먼저 호출한 다음 드로잉 메서드 중 하나를 호출하십시오. 이 과정은 종이에서 펜을 들어 다른 위치로 옮기는 것에 견줄 수 있습니다.

드로잉 포인트를 지정했으면 드로잉 메서드인 `lineTo()`(직선 그리기용) 및 `curveTo()`(곡선 그리기용)를 연속적으로 호출하여 그리기를 수행하십시오.

팁 **그리는 동안 언제든지 `moveTo()` 메서드를 호출하여 드로잉 포인트를 다른 새로운 위치로 이동할 수 있습니다.**

그리기 단계에서 채움 색상을 지정한 경우, `endFill()` 메서드를 호출하여 해당 채우기를 종료하도록 Adobe Flash Player에 지시할 수 있습니다. 닫힌 모양 그리기가 아닌 경우(즉 `endFill()`을 호출했을 때 드로잉 포인트가 모양의 시작점에 있지 않은 경우), `endFill()` 메서드를 호출하면 Flash Player가 현재 드로잉 포인트에서 가장 최근의 `moveTo()` 호출에 지정된 위치까지 직선을 그려 모양을 자동으로 닫습니다. 채우기를 시작했는데 `endFill()`을 호출하지 않은 경우, `beginFill()`(또는 다른 채우기 메서드 중 하나)을 호출하면 현재 채우기가 종료되고 새로운 채우기가 시작됩니다.

직선 그리기

`lineTo()` 메서드를 호출하면 Graphics 객체는 현재 드로잉 포인트에서 메서드 호출에서 두 개의 매개 변수로 지정한 좌표까지 지정한 선 스타일을 사용하여 직선을 그립니다. 예를 들어, 다음 코드 행은 드로잉 포인트를 100, 100 좌표에 두고 200, 200 좌표까지 선을 그립니다.

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

다음 예제에서는 높이가 100픽셀인 빨강과 녹색 삼각형을 그립니다.

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// 0, 0 좌표에서 시작하는 빨강 삼각형
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight/2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight/2, 0);
```

```
// 200, 0 좌표에서 시작하는 녹색 삼각형
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight/2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight/2, 0);

this.addChild(triangle);
```

곡선 그리기

curveTo() 메서드는 이차 베지어 곡선을 그립니다. 이 곡선은 두 개의 포인트(앵커 포인트)를 연결하는 호를 그리며 이 호는 세 번째 포인트(제어 포인트) 방향으로 구부러집니다. Graphics 객체는 현재 드로잉 위치를 첫 번째 앵커 포인트로 사용합니다. curveTo() 메서드를 호출하면 4개의 매개 변수(제어 포인트의 x, y 좌표와 두 번째 앵커 포인트의 x, y 좌표)가 전달됩니다. 예를 들어, 다음 코드는 100, 100 포인트에서 시작하여 200, 200 포인트에서 끝나는 곡선을 그립니다. 제어 포인트가 175, 125이므로 오른쪽으로 이동하여 아래로 향하는 곡선이 만들어집니다.

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

다음 예제에서는 폭 및 높이가 100픽셀인 빨강과 녹색 원형 객체를 그립니다. 이차 베지어 수식의 특성상 완벽한 원이 되지는 않습니다.

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// 빨강 원 모양
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// 녹색 원 모양
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

내장 메서드를 사용하여 모양 그리기

ActionScript 3.0에는 원, 타원, 사각형, 모서리가 둥근 사각형 등의 일반 모양을 쉽게 그릴 수 있는 몇 가지 메서드가 포함되어 있습니다. Graphics 클래스의 `drawCircle()`, `drawEllipse()`, `drawRect()`, `drawRoundRect()` 및 `drawRoundRectComplex()` 메서드가 이에 해당합니다. 이러한 메서드는 `lineTo()` 및 `curveTo()` 메서드 대신 사용할 수 있습니다. 하지만 이러한 메서드를 호출하기 전에 선 및 채우기 스타일도 지정해야 합니다.

다음 예제에서는 폭과 너비가 100픽셀인 빨강, 녹색 및 파랑 사각형을 그리는 예제를 다시 구성한 것입니다. 다음 코드는 `drawRect()` 메서드를 사용하고 채우기 색상의 알파를 50%(0.5)로 지정합니다.

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

Sprite 또는 MovieClip 객체에서 `graphics` 속성으로 만든 드로잉 내용은 항상 해당 객체에 포함된 모든 자식 표시 객체의 뒤에 나타납니다. 또한 `graphics` 속성 내용은 별도의 표시 객체가 아니므로 Sprite 또는 MovieClip 객체의 자식 목록에 나타나지 않습니다. 예를 들어 다음 Sprite 객체는 `graphics` 속성을 사용하여 그린 원으로, 해당 자식 표시 객체 목록에 TextField 객체가 있습니다.

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

TextField는 graphics 객체로 그린 원형의 위쪽에 표시됩니다.

그래디언트 선 및 채우기 만들기

graphics 객체는 또한 단색보다 그래디언트를 사용하여 획 및 채우기를 그릴 수 있습니다. 그래디언트 획은 `lineGradientStyle()` 메서드를 사용하여 만들고, 그래디언트 채우기는 `beginGradientFill()` 메서드를 사용하여 만듭니다.

두 메서드 모두 동일한 매개 변수를 사용합니다. 처음 4개의 매개 변수인 유형, 색상, 알파, 비율은 필수 항목입니다. 나머지 4개의 매개 변수는 선택 사항이지만 고급 사용자 정의 시 유용합니다.

- 첫 번째 매개 변수는 만들려는 그래디언트 유형을 지정하며 사용할 수 있는 값은 `GradientFill.LINEAR` 또는 `GradientFill.RADIAL`입니다.
- 두 번째 매개 변수는 사용할 색상 값의 배열을 지정합니다. 선형 그래디언트에서는 색상이 왼쪽에서 오른쪽으로 배열되고 방사형 그래디언트에서는 안쪽에서 바깥쪽으로 배열됩니다. 배열 색상의 순서는 그래디언트에서 색상이 그려지는 순서를 나타냅니다.
- 세 번째 매개 변수는 이전 매개 변수의 해당 색상에 대한 알파 투명도 값을 지정합니다.
- 네 번째 매개 변수는 비율, 즉 각 색상이 그래디언트에서 가지는 강도를 지정합니다. 사용할 수 있는 값은 0에서 255까지입니다. 이러한 값은 폭이나 높이를 나타내는 것이 아니라 그래디언트에서의 위치를 나타냅니다. 즉, 0은 그래디언트의 처음을 나타내고 255는 그래디언트의 끝을 나타냅니다. 비율의 배열은 순차적으로 증가해야 하며 두 번째 및 세 번째 매개 변수에 지정된 색상 및 알파 배열과 동일한 항목 수를 가져야 합니다.

다섯 번째 매개 변수인 변형 행렬은 선택 사항이지만 간단한 방법으로 그래디언트의 모양을 효율적으로 제어할 수 있으므로 흔히 사용됩니다. 이 매개 변수는 Matrix 인스턴스를 사용합니다. 그래디언트에 대한 Matrix 객체를 만드는 가장 쉬운 방법은 Matrix 클래스의 `createGradientBox()` 메서드를 사용하는 것입니다.

그래디언트에 사용할 Matrix 객체 정의

flash.display.Graphics 클래스의 `beginGradientFill()` 및 `lineGradientStyle()` 메서드를 사용하면 모양에 사용할 그래디언트를 정의할 수 있습니다. 그래디언트를 정의하는 경우 행렬을 이러한 메서드의 매개 변수 중 하나로 제공합니다.

행렬을 정의하는 가장 쉬운 방법은 Matrix 클래스의 `createGradientBox()` 메서드를 사용하여 그래디언트 정의에 사용되는 행렬을 만드는 것입니다. `createGradientBox()` 메서드에 전달되는 매개 변수를 사용하여 그래디언트의 크기, 회전 및 위치를 정의합니다. `createGradientBox()` 메서드는 다음과 같은 매개 변수를 사용합니다.

- 그래디언트 상자 폭: 그래디언트가 펼쳐지는 폭(단위: 픽셀)
- 그래디언트 상자 높이: 그래디언트가 펼쳐지는 높이(단위: 픽셀)
- 그래디언트 상자 회전: 그래디언트에 적용될 회전(단위: 라디안)
- 수평 이동: 그래디언트가 수평으로 이동되는 정도(단위: 픽셀)

- 수직 이동: 그래디언트가 수직으로 이동되는 정도(단위: 픽셀)

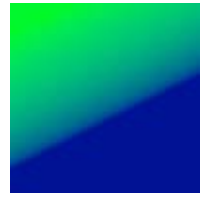
예를 들어 다음과 같은 특성을 갖는 그래디언트를 검토하십시오.

- GradientType.LINEAR
- ratios 배열이 [0, 255]로 설정된 녹색과 파랑의 두 색상
- SpreadMethod.PAD
- InterpolationMethod.LINEAR_RGB

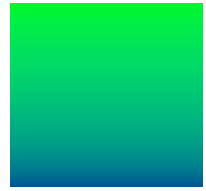
다음 예제에서는 createGradientBox() 메서드의 rotation 매개 변수만 다르고 다른 모든 설정은 동일한 그래디언트를 보여 줍니다.

```
width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```

```
width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
```

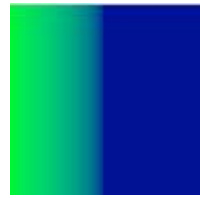


```
width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```



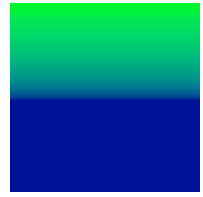
다음 예제에서는 녹색에서 파란색으로 바뀌는 선형 그래디언트 효과를 보여 줍니다. 이 때 `createGradientBox()` 메서드의 `rotation`, `tx` 및 `ty` 매개 변수만 다르고 다른 모든 설정은 동일합니다.

```
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```

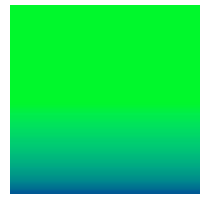


```
width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;
```

```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

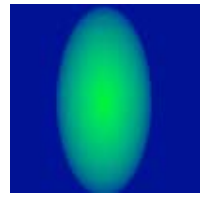


```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;
```



다음 예제와 같이 `createGradientBox()` 메서드의 `width`, `height`, `tx` 및 `ty` 매개 변수는 *방사형* 그래디언트 채우기의 크기와 위치에도 영향을 줍니다.

```
width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
```



다음 코드는 마지막에 보여 준 방사형 그래디언트를 만듭니다.

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                colors,
                alphas,
                ratios,
                matrix,
                spreadMethod,
                interp,
                focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

그래디언트 채우기의 폭과 높이는 Graphics 객체로 그린 폭이나 높이가 아닌 그래디언트 행렬의 폭과 높이에 따라 결정됩니다. Graphics 객체를 사용하여 그리면 그래디언트 행렬의 해당 좌표에 존재하는 객체가 그려집니다. Graphics 객체의 shape 메서드 중 하나(예: `drawRect()`)를 사용하더라도 그려지는 모양의 크기까지 그래디언트를 확장할 수는 없습니다. 그래디언트의 크기는 그래디언트 행렬 자체에 지정해야 합니다.

다음을 통해 그래디언트 행렬의 차원과 드로잉 자체의 차원 간에 존재하는 시각적 차이를 확인할 수 있습니다.

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000,
  0x00FF00, 0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
```

```
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

이 코드는 빨강, 녹색, 파랑이 똑같이 배분된 동일한 채우기 스타일을 사용하여 세 가지 그래디언트를 그립니다. 그래디언트는 픽셀 폭이 각각 50, 100, 150인 `drawRect()` 메서드를 사용하여 그려집니다. `beginGradientFill()` 메서드에 지정된 그래디언트 행렬은 100픽셀의 폭을 사용하여 만들어집니다. 즉, 첫 번째 그래디언트에는 그래디언트 스펙트럼의 반만 포함되고, 두 번째 그래디언트에는 스펙트럼 전체가 포함되며, 세 번째 그래디언트에는 스펙트럼 전체를 포함하면서 추가적으로 50픽셀의 파랑이 오른쪽으로 확장됩니다.

`lineGradientStyle()` 메서드는 그래디언트를 정의하는 것 외에는 `beginGradientFill()`과 유사하게 동작하지만 그리기 전에 `lineStyle()` 메서드를 사용하여 획 두께를 지정해야 합니다. 다음 코드는 빨강, 녹색 및 파랑 그래디언트 획을 가진 상자를 그립니다.

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000,
  0x00FF00, 0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Matrix 클래스에 대한 자세한 내용은 414페이지의 “Matrix 객체 사용”을 참조하십시오.

드로잉 메서드와 Math 클래스 사용

Graphics 객체는 원과 사각형을 그리지만 특히 드로잉 메서드를 Math 클래스의 속성 및 메서드와 함께 사용할 경우 보다 복잡한 형태를 그릴 수도 있습니다. Math 클래스에는 일반적인 수학적 비율을 나타내는 상수가 포함되어 있습니다. 예를 들어, 원주율에 대한 상수인 `Math.PI` (약 3.14159265...)가 이에 해당합니다. 또한 `Math.sin()`, `Math.cos()` 및 `Math.tan()` 등의 삼각 함수에 대한 메서드도 포함합니다. 이러한 메서드 및 상수를 사용하여 모양을 그리면 특히 반복 또는 재귀와 함께 사용할 경우 더욱 동적인 시각 효과를 만들 수 있습니다.

Math 클래스의 많은 메서드는 원 치수를 각도 단위가 아닌 라디안 단위로 예상하므로, Math 클래스의 일반적인 용도는 이 두 유형의 단위를 상호 변환하는 것입니다.

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

다음 예제에서는 사인파와 코사인파를 만들어 특정한 값에 대한 `Math.sin()` 메서드와 `Math.cos()` 메서드의 차이점을 강조 표시합니다.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
```

```
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
  var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
  var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

  graphics.beginFill(sinWaveColor);
  graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
  graphics.beginFill(cosWaveColor);
  graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

드로잉 API를 사용한 애니메이션

드로잉 API를 사용하여 내용을 만들면 내용을 한 번만 배치하도록 제한되지 않는다는 장점이 있습니다. 그릴 때 사용하는 변수를 유지하거나 수정하여 드로잉 내용을 수정할 수 있습니다. 특정 프레임 기간 동안 또는 타이머를 사용하여 변수를 변경하거나 다시 그림으로써 애니메이션을 전달할 수 있습니다.

예를 들어, 다음 코드는 `Event.ENTER_FRAME` 이벤트 수신을 통해 각 전달 프레임으로 표시를 변경하고 현재 각도를 증가시켜 graphics 객체를 지우고 업데이트된 위치에서 다시 그리도록 지시합니다.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
  currentDegrees += 4;
  var radians:Number = getRadians(currentDegrees);
  var posX:Number = Math.sin(radians) * radius;
  var posY:Number = Math.cos(radians) * radius;
```

```
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
    return degrees * Math.PI / 180;
}
```

현저히 다른 결과를 얻기 위한 실험으로 `currentDegrees`, `radius`, `satelliteRadius` 및 해당 코드의 시작 부분에 있는 초기 난수 변수를 수정해 볼 수 있습니다. 예를 들어, radius 변수를 감소시키거나 totalSatellites 변수를 증가시켜 봅니다. 다음은 드로잉 API를 시각적으로 표시하는 방법을 보여 주는 예제로서, 시각적으로는 복잡하지만 생성 원리는 단순합니다.

예제: Algorithmic Visual Generator

Algorithmic Visual Generator 예제에서는 원 궤도를 따라 움직이는 여러 가지 "위성" 또는 원형을 스테이지에 동적으로 그립니다. 살펴본 기능은 다음과 같습니다.

- 드로잉 API를 사용하여 동적 모양을 가진 기본 모양 그리기
- 드로잉에 사용된 속성과 사용자 상호 작용 연결
- 각 프레임에서 스테이지를 지우고 다시 그려서 애니메이션 전달

이전 하위 섹션의 예제에서는 `Event.ENTER_FRAME` 이벤트를 사용하여 단일 "위성" 또는 애니메이션을 적용했습니다. 이 예제에서는 이를 바탕으로 여러 위성의 시각적 표시를 즉시 업데이트하는 각종 슬라이더의 제어판을 만듭니다. 이 예제는 코드를 외부 클래스로 정형화하고 위성 생성 코드를 루프 내에 포함시켜 각 위성에 대한 참조를 `satellites` 배열에 저장합니다.

이 샘플에 대한 응용 프로그램 파일을 구하려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 방문하십시오. 응용 프로그램 파일은 Samples/AlgorithmicVisualGenerator 폴더에 있습니다. 이 폴더에는 다음과 같은 파일이 있습니다.

파일	설명
AlgorithmicVisualGenerator.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	스테이지에 대한 위성을 드로잉하고 제어판의 이벤트에 응답하여 위성 드로잉에 영향을 미치는 변수를 업데이트하는 등의 기본 응용 프로그램 기능을 제공하는 클래스입니다.

파일	설명
com/example/programmingas3/algorithmic/ControlPanel.as	몇 개의 슬라이더로 사용자 상호 작용을 관리하고 상호 작업 발생 시 이벤트를 전달하는 클래스입니다.
com/example/programmingas3/algorithmic/Satellite.as	중앙 포인트 주위를 궤도로 회전하는 표시 객체를 나타내고 현재 드로잉 상태와 관련된 속성을 포함하는 클래스입니다.

리스너 설정

응용 프로그램은 먼저 세 개의 리스너를 만듭니다. 첫 번째 리스너는 제어판에서 전달 이벤트를 수신하며 위성 재구성이 필요합니다. 두 번째 리스너는 SWF 파일의 스테이지 크기 변경 사항을 수신합니다. 세 번째 리스너는 SWF 파일의 각 전달 프레임을 수신하고 `doEveryFrame()` 함수를 사용하여 그리기를 다시 수행합니다.

위성 만들기

이러한 리스너가 설정되면 `build()` 함수가 호출됩니다. 이 함수는 먼저 `clear()` 함수를 호출하여 `satellites` 배열을 비우고 스테이지에 대한 이전 드로잉을 모두 지웁니다. 제어판이 이와 같은 이벤트를 보낼 때마다(예를 들어 색상 설정이 변경될 때마다) `build()` 함수를 다시 호출할 수 있으므로 이 과정이 필요합니다. 이 경우 위성을 제거하고 다시 만들어야 합니다.

그런 다음 함수는 생성에 필요한 초기 속성(예: 궤도의 임의 위치에서 시작하는 `position` 변수, 이 예제에서 위성이 생성되면 변경되지 않는 `color` 변수)을 설정하여 위성을 생성합니다.

각 위성이 생성되었으면 해당 참조가 `satellites` 배열에 추가됩니다. `doEveryFrame()` 함수가 호출되면 해당 함수가 이 배열의 모든 위성에 대해 업데이트됩니다.

위성 위치 업데이트

`doEveryFrame()` 함수는 응용 프로그램의 애니메이션 프로세스의 핵심 요소로서, SWF 파일의 프레임 속도와 동일한 속도로 각 프레임마다 호출됩니다. 그리기 변수는 약간씩 달라지므로 이 함수는 애니메이션 모양을 전달합니다.

함수는 먼저 이전 드로잉을 모두 지우고 배경을 다시 그립니다. 그런 다음 각 위성 컨테이너를 반복하여 각 위성의 `position` 속성을 증가시키고 제어판의 사용자 상호 작용으로 인해 변경되었을 수 있는 `radius` 및 `orbitRadius` 속성을 업데이트합니다. 마지막으로 위성은 Satellite 클래스의 `draw()` 메서드를 호출하여 새로운 위치로 업데이트됩니다.

카운터 `i`는 `visibleSatellites` 변수까지만 증가됩니다. 제어판을 통해 표시되는 위성의 수를 사용자가 제한한 경우 루프의 나머지 위성은 다시 그려지지 않고 숨겨져야 하기 때문입니다. 이러한 경우는 그리기를 담당하는 루프 바로 다음의 루프에서 발생합니다.

doEveryFrame() 함수가 완료되면 `visibleSatellites` 수가 화면 위치에서 업데이트됩니다.

사용자 상호 작용에 대한 응답

사용자 상호 작용은 제어판을 통해 발생하며 ControlPanel 클래스에서 관리합니다. 이 클래스는 각 슬라이더의 개별적인 최소값, 최대값 및 기본값과 함께 리스너를 설정합니다. 사용자가 이러한 슬라이더를 이동하면 `changeSetting()` 함수가 호출되며 이 함수는 제어판의 속성을 업데이트합니다. 변경 사항으로 인해 표시를 재구성해야 하는 경우 이벤트가 전달되어 기본 응용 프로그램 파일에서 처리됩니다. 제어판 설정이 변경되면 `doEveryFrame()` 함수는 업데이트된 변수를 사용하여 각 위성을 그립니다.

구체적인 사용자 정의

이 예제는 드로잉 API를 사용하여 시각적 표시 생성 방법을 보여 주는 간단한 예제로서, 비교적 간단한 코드 행을 사용하여 꽤 복잡해 보이는 대화형 환경을 만듭니다. 하지만 이 예제를 약간 수정하여 기능을 추가할 수 있습니다. 몇 가지 아이디어는 다음과 같습니다.

- `doEveryFrame()` 함수는 위성의 색상 값을 증가시킬 수 있습니다.
- `doEveryFrame()` 함수는 시간 경과에 따라 위성 반경을 축소하거나 확장할 수 있습니다.
- 위성 반경은 원형일 필요가 없습니다. 예를 들어, Math 클래스를 사용하여 사인파에 따라 이동할 수 있습니다.
- 위성은 다른 위성과의 히트 감지를 사용할 수 있습니다.

드로잉 API를 사용하여 Flash 제작 환경에서 시각 효과를 생성함으로써 런타임에 기본 모양을 그릴 수도 있습니다. 하지만 손으로 제작할 수 없는 다양하고 넓은 범위의 시각 효과를 생성하는 데도 사용될 수 있습니다. ActionScript 제작자는 드로잉 API와 약간의 수학 기능을 사용하여 여러 가지 다양한 작품에 생동감을 불어 넣을 수 있습니다.

제 15 장

표시 객체 필터링

15

이전에는 비트맵 이미지에 필터 효과를 적용하는 것은 Adobe Photoshop®과 Adobe Fireworks® 등의 특수한 이미지 편집 소프트웨어 영역으로 한정되어 있었습니다. ActionScript 3.0에는 여러 비트맵 효과 필터 클래스를 포함하여 개발자가 프로그래밍 방식으로 비트맵과 표시 객체에 필터를 적용하여 그래픽 조작 응용 프로그램에서 사용할 수 있는 것과 동일한 효과를 낼 수 있는 flash.filters 패키지가 들어 있습니다.

목차

표시 객체 필터링의 기초

표시 객체 필터링 소개

응용 프로그램을 한층 더 세련되게 만드는 방법 중 하나는 사진 뒤에 그림자를 추가하여 3차원 효과를 내거나 버튼 주위에 빛 효과를 주어 해당 버튼이 활성화되어 있음을 알리는 등 간단한 그래픽 효과를 추가하는 것입니다. ActionScript 3.0에서는 모든 표시 객체 또는 BitmapData 인스턴스에 적용 가능한 9개의 필터를 제공하는데, 여기에는 그림자 및 광선 필터 등과 같은 기본 필터에서 위치 변경 맵 필터 및 회선 필터 등 다양한 효과를 내는 복잡한 필터가 포함됩니다.

일반적인 필터링 작업

ActionScript에서 필터를 사용하여 수행할 수 있는 작업은 다음과 같습니다.

- 필터 만들기
- 표시 객체에 필터 적용
- BitmapData 인스턴스의 이미지 데이터에 필터 적용
- 객체에서 필터 제거
- 다음과 같은 다양한 필터 효과 만들기
  - 광선
  - 흐림
  - 그림자
  - 선명도
  - 위치 변경
  - 가장자리 감지
  - 엠보싱
  - 기타 효과

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 경사: 두 면의 픽셀은 밝게 하고 반대편 두 면의 픽셀은 어둡게 하여 만들어진 가장자리로, 위로 솟은 버튼 또는 안으로 살짝 들어간 버튼이나 그와 유사한 그래픽을 만들기 때 자주 사용되는 3차원 테두리 효과를 연출합니다.
- 회선: 다양한 비율을 적용하여 각 픽셀의 값을 인접한 픽셀 중 일부 또는 전부의 값과 결합하여 이미지의 픽셀을 왜곡시키는 것입니다.
- 위치 변경: 이미지의 픽셀을 새 위치로 옮기는 것입니다.
- 행렬: 격자 수를 여러 값에 적용한 다음 결과를 결합하여 특정 수학적 계산을 수행할 때 사용하는 숫자 배열입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장에서는 시각적 내용의 작성 및 조작을 다루므로 코드 목록을 테스트하려면 해당 코드를 실행한 후 작성된 SWF 파일에서 결과를 확인해야 합니다. 대부분의 예제에서는 드로잉 API를 사용하여 내용을 작성하거나 적용 대상 필터에 이미지를 로드합니다.

이 장의 코드를 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드를 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   작성된 SWF 파일에서 코드의 결과를 확인합니다.

대부분의 예제 코드에는 비트맵 이미지를 작성하는 코드가 포함되어 있기 때문에 비트맵 내용을 제공하지 않고도 직접 코드를 테스트할 수 있습니다. 또한 코드 샘플을 변경하여 자체 이미지에 로드하고 예제의 이미지 대신 사용할 수 있습니다.

필터 작성 및 적용

필터를 사용하면 비트맵 및 표시 객체에 그림자, 경사 및 흐림 효과 등 다양한 효과를 적용할 수 있습니다. 각 필터는 클래스로 정의되므로 필터를 적용하면 필터 객체의 인스턴스가 기타 다른 객체가 생성되는 것과 마찬가지 방식으로 만들어집니다. 일단 filter 객체를 만든 후에는 해당 필터를 객체의 `filters` 속성이나, BitmapData 객체의 경우 `applyFilter()` 메서드를 사용하여 간단하게 표시 객체에 적용할 수 있습니다.

새 필터 작성

새 필터 객체를 만들려면 선택한 필터 클래스의 생성자 메서드를 호출하면 됩니다. 예를 들어 DropShadowFilter 객체를 만들려면 다음 코드를 사용하십시오.

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

여기에 표시되어 있지는 않지만 모든 filter 클래스 생성자 등의 `DropShadowFilter()` 생성자는 필터 효과 모양 사용자 정의에 사용할 수 있는 여러 선택적 매개 변수를 사용합니다.

필터 적용

생성한 필터 객체는 표시 객체 또는 BitmapData 객체에 적용할 수 있습니다. 필터 적용 방법은 필터를 적용하는 객체에 따라 달라집니다.

표시 객체에 필터 적용

표시 객체에 필터 효과를 적용하는 경우에는 `filters` 속성을 통해 적용합니다. 표시 객체의 `filters` 속성은 Array 인스턴스이며, 그 요소는 표시 객체에 적용된 filter 객체입니다. 표시 객체에 단일 필터를 적용하려면 다음과 같이 filter 인스턴스를 만들고 이를 Array 인스턴스에 추가한 다음 표시 객체의 `filters` 속성에 Array 객체를 지정합니다.

```
import flash.display.Bitmap;
```

```
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// bitmapData 객체를 만들어서 화면에 렌더링합니다 .
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// DropShadowFilter 인스턴스를 만듭니다 .
var dropShadow:DropShadowFilter = new DropShadowFilter();

// 필터를 Array() 생성자에 매개 변수로 전달하여 배열에 필터를 추가하는
// 방식으로 필터 배열을 만듭니다 .
var filtersArray:Array = new Array(dropShadow);

// 필터를 적용할 표시 객체에 필터 배열을 지정합니다 .
myDisplayObject.filters = filtersArray;
```

객체에 여러 필터를 지정하려면 `filters` 속성에 Array 인스턴스를 지정하기 전에 먼저 모든 필터를 해당 인스턴스에 추가합니다. Array 생성자에 여러 객체를 매개 변수로 전달하면 해당 객체를 Array에 추가할 수 있습니다. 예를 들어 다음 코드는 이전에 만든 표시 객체에 경사 필터 및 광선 필터를 적용합니다.

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// 필터를 만들어서 배열에 추가합니다 .
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// 필터를 적용할 표시 객체에 필터 배열을 지정합니다 .
myDisplayObject.filters = filtersArray;
```

예제

필터를 포함하는 배열을 만드는 경우에는 이전 예제와 같이 `new Array()` 생성자를 사용하여 만들거나 Array 리터럴 구문을 사용하여 해당 필터를 각괄호(`[]`)로 묶습니다. 다음 코드 행을 예로 들어 보겠습니다.

```
var filters:Array = new Array(dropShadow, blur);
```

위 코드 행은 아래 코드 행과 동일한 작업을 수행합니다.

```
var filters:Array = [dropShadow, blur];
```

표시 객체에 여러 개의 필터를 적용할 경우 필터는 누적형의 순차적인 방식으로 적용됩니다. 예를 들어 두 개의 요소가 있는 필터 배열의 경우 먼저 경사 필터 그 다음으로 그림자 필터가 추가되며, 그림자 필터는 경사 필터와 표시 객체 모두에 적용됩니다. 그 이유는 그림자 필터가 필터 배열에서 두 번째 위치에 있기 때문입니다. 비누적 방식으로 필터를 적용하려면 표시 객체의 새 복사본에 각각의 필터를 적용해야 합니다.

표시 객체에 한 개 또는 소수의 필터만 지정하려는 경우 필터 인스턴스를 만든 다음 단일 명령문을 사용하여 객체에 해당 인스턴스를 지정할 수 있습니다. 예를 들어, 다음 코드 행은 흐림 필터를 myDisplayObject라는 표시 객체에 적용합니다.

```
myDisplayObject.filters = [new BlurFilter()];
```

이전 코드는 Array 리터럴 구문(각괄호)을 사용하여 Array 인스턴스를 만들고 Array의 요소로 새 BlurFilter 인스턴스를 만든 다음 해당 Array를 myDisplayObject라는 표시 객체의 filters 속성에 지정합니다.

표시 객체에서 필터 제거

표시 객체에서 모든 필터를 제거하려면 다음과 같이 단순히 filters 속성에 null 값을 지정하면 됩니다.

```
myDisplayObject.filters = null;
```

객체에 여러 객체를 적용한 경우 하나의 필터만 제거하려면 여러 단계를 거쳐 filters 속성 배열을 변경해야 합니다. 자세한 내용은 442페이지의 “런타임에 필터 변경”을 참조하십시오.

BitmapData 객체에 필터 적용

BitmapData 객체에 필터를 적용하려면 BitmapData 객체의 applyFilter() 메서드를 사용해야 합니다.

```
myBitmapData.applyFilter(sourceBitmapData);
```

applyFilter() 메서드는 필터를 BitmapData 객체에 적용하여 필터링된 새 이미지를 생성합니다. 이 메서드는 원본 소스 이미지를 수정하지 않는 대신 소스 이미지에 적용된 필터 결과가 applyFilter() 메서드를 호출하는 BitmapData 인스턴스에 저장됩니다.

필터 작동 방법

표시 객체 필터링은 원본 객체의 복사본을 투명 비트맵으로 캐싱하는 방식으로 작동됩니다.

표시 객체에 필터를 적용한 후에는 Adobe Flash Player에서 해당 객체에 유효한 필터 목록이 있는 한 해당 객체를 비트맵으로 캐시합니다. 이 소스 비트맵은 이후에 적용되는 모든 필터 효과의 원본 이미지로 사용됩니다.

일반적으로 각 표시 객체에는 두 개의 비트맵이 포함되어 있는데, 한 개는 필터링되지 않은 원본 소스 표시 객체에 대한 비트맵이고 다른 한 개는 필터링 이후의 최종 이미지에 대한 비트맵입니다. 최종 이미지는 렌더링 시 사용되며 표시 객체가 변경되지 않으면 업데이트할 필요가 없습니다.

필터 작업의 잠재적 문제점

필터를 사용하여 작업할 경우 혼동하거나 문제가 발생되지 않도록 유념해야 할 몇 가지 사항이 있습니다. 이에 대해서는 다음 단원에서 설명합니다.

필터 및 비트맵 캐싱

표시 객체에 필터를 적용하기 위해서는 해당 객체에 대한 비트맵 캐싱이 활성화되어야 합니다. `cacheAsBitmap` 속성이 `false`로 설정된 표시 객체에 필터를 적용하는 경우 Flash Player는 자동으로 해당 객체의 `cacheAsBitmap` 속성 값을 `true`로 설정합니다. 나중에 표시 객체에서 모든 필터를 제거하면 `cacheAsBitmap` 속성이 마지막 설정된 값으로 재설정됩니다.

런타임에 필터 변경

표시 객체에 이미 하나 이상의 필터가 적용된 경우에는 `filters` 속성 배열에 필터를 추가할 수 없습니다. 적용한 필터 집합을 추가 또는 변경하려면 전체 필터 배열 복사본을 만들어 이 (임시) 배열을 수정합니다. 그런 다음 이 배열을 표시 객체의 `filters` 속성에 다시 지정하여 객체에 해당 필터를 적용합니다. 다음 코드는 이 과정을 보여 줍니다. 처음에는 광선 필터가 `myDisplayObject`라는 표시 객체에 적용되고 나중에 해당 표시 객체를 클릭하면 `addFilters()` 함수가 호출됩니다. 이 함수에서 다음과 같이 두 개의 추가 필터가 myDisplayObject에 적용됩니다.

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
  // 필터 배열을 복사합니다 .
  var filtersCopy:Array = myDisplayObject.filters;

  // 필터를 원하는 대로 변경합니다 . 이 경우 필터를 추가합니다 .
  filtersCopy.push(new BlurFilter());
  filtersCopy.push(new DropShadowFilter());

  // filters 속성에 배열을 다시 지정하여 변경 내용을 적용합니다 .
  myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

필터 및 객체 변형

표시 객체의 경계 상자 사각형 외부의 필터링되지 않은 영역(예: 그림자)은 인스턴스가 다른 인스턴스와 겹치거나 교차하는지 여부를 판단하는 히트 감지를 목적으로 하는 표면의 일부로 간주됩니다. DisplayObject 클래스의 히트 감지 메서드는 벡터에 기반을 두기 때문에 비트맵 결과에 대해서는 히트 감지를 수행할 수 없습니다. 예를 들어, 버튼 인스턴스에 경사 필터를 적용하는 경우 경사가 적용된 인스턴스 부분에 대해서는 히트 감지를 사용할 수 없습니다.

필터에서는 크기 조절, 회전 및 기울이기를 지원하지 않습니다. 필터링된 표시 객체 자체의 크기가 조절되는 경우(`scaleX` 및 `scaleY`가 100%가 아닌 경우)에는 인스턴스에 필터 효과가 적용될 때 크기가 조절되지 않습니다. 다시 말해서 인스턴스의 원래 모양에만 회전, 크기 조절 또는 기울이기가 수행되며, 필터가 적용된 인스턴스에는 회전, 크기 조절 또는 기울이기가 수행되지 않습니다.

실제 효과 또는 중첩 인스턴스를 만드는 필터를 사용하여 인스턴스에 애니메이션을 적용하고 BitmapData 클래스를 사용하여 이 효과를 나타내도록 필터에 애니메이션을 적용할 수 있습니다.

Filter 및 Bitmap 객체

BitmapData 객체에 필터를 적용하면 `cacheAsBitmap` 속성이 자동으로 `true`로 설정됩니다. 실제로는 이런 방식을 사용하여 필터가 원본이 아닌 객체의 복사본에 적용됩니다.

그런 다음 이 복사본은 가장 가까운 픽셀과 최대한 가깝게 기본 디스플레이의 원본 객체 위에 배치됩니다. 원본 비트맵 경계가 변경되면 필터링된 비트맵 복사본은 확장되거나 왜곡되지 않고 다시 만들어집니다.

표시 객체의 모든 필터를 지우면 `cacheAsBitmap` 속성이 필터 적용 이전 상태로 재설정됩니다.

사용 가능한 표시 필터

ActionScript 3.0에는 다음과 같이 표시 객체와 BitmapData 객체에 적용할 수 있는 9개의 필터 클래스가 들어 있습니다.

- 경사 필터(BevelFilter 클래스)
- 흐림 필터(BlurFilter 클래스)
- 그림자 필터(DropShadowFilter 클래스)
- 광선 필터(GlowFilter 클래스)
- 그래디언트 경사 필터(GradientBevelFilter 클래스)
- 그래디언트 광선 필터(GradientGlowFilter 클래스)
- 색상 매트릭스 필터(ColorMatrixFilter 클래스)
- 회선 필터(ConvolutionFilter 클래스)

■ 위치 변경 맵 필터(DisplacementMapFilter 클래스)

처음 여섯 개 필터는 하나의 특정 효과를 만들 때 사용할 수 있는 간단한 필터로 몇 가지 효과 사용자 정의 기능을 제공합니다. 이 여섯 가지 필터는 ActionScript를 사용하여 적용할 수 있으며 필터 패널을 사용하여 Adobe Flash CS3 Professional의 객체에도 적용할 수 있습니다. 따라서 ActionScript를 사용하여 필터를 적용하는 경우라도 Flash 제작 도구를 통해 시각 인터페이스에서 여러 가지 필터 및 설정을 신속하게 시험하여 원하는 효과를 얻는 방법을 알 수 있습니다.

마지막 세 필터는 ActionScript에서만 사용할 수 있는 필터입니다. 색상 매트릭스 필터, 회선 필터 및 위치 변경 맵 필터는 여러 가지 유형의 효과를 만드는 데 사용할 수 있으며 하나의 효과를 내기 위해 최적화되었다기보다는 강력한 기능과 유연성을 제공합니다. 예를 들어 여러 매트릭스 값을 선택함으로써 흐리게 하기, 엠보싱, 선명하게 하기, 색상 가장자리 찾기, 변형 등의 효과를 내는 데 회선 필터를 사용할 수 있습니다.

간단한 필터 또는 복잡한 필터 모두 각각 해당 속성을 사용하여 사용자 정의할 수 있습니다. 필터 속성은 일반적으로 두 가지 방법을 통해 설정할 수 있습니다. 모든 필터의 속성은 필터 객체의 생성자에 매개 변수 값을 전달하여 설정할 수 있습니다. 또는 매개 변수 전달을 통해 필터 속성을 설정하는지 여부와 상관없이 필터 객체의 속성 값을 설정하여 나중에 해당 필터를 조정할 수 있습니다. 대부분의 예제 코드 샘플은 예제를 쉽게 따라할 수 있도록 속성을 직접 설정하고 있습니다. 그러나 대개 필터 객체 생성자에 값을 매개 변수로 전달하면 보다 간단한 코드로 동일한 결과를 얻을 수 있습니다. 각 필터, 필터 속성 및 생성자 매개 변수에 대한 보다 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 flash.filters 패키지에 대한 항목을 참조하십시오.

경사 필터

BevelFilter 클래스는 필터가 적용된 객체에 경사진 3D 가장자리를 추가할 수 있도록 지원합니다. 이 필터는 객체의 단단한 모서리 또는 가장자리가 옆으로 경사진 것처럼 보이게 만듭니다.

BevelFilter 클래스 속성을 사용하여 경사 모양을 사용자 정의할 수 있습니다. 즉, 강조 색상 및 그림자 색상을 설정할 수 있으며 경사 가장자리 흐리게 하기, 경사 각도, 경사 가장자리 위치 등을 설정할 수 있으며 녹아웃 효과를 연출할 수도 있습니다.

다음 예제에서는 외부 이미지를 로드하여 경사 필터를 적용합니다.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// 이미지를 스테이지로 로드합니다 .
var imageLoader:Loader = new Loader();
```

```
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// 경사 필터를 만들고 filter 속성을 설정합니다 .
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// 이미지에 필터를 적용합니다 .
imageLoader.filters = [bevel];
```

흐림 필터

BlurFilter 클래스는 표시 객체 및 객체의 내용을 칠하거나 흐리게 합니다. 흐림 효과는 객체가 초점을 벗어나 있다는 인상을 심거나 모션 흐림에서처럼 빠른 움직임을 시뮬레이트하는 데 유용합니다. 흐림 필터의 `quality` 속성을 낮음으로 설정하면 부드럽게 흐린 렌즈 효과를 시뮬레이션할 수 있습니다. `quality` 속성을 높음으로 설정하면 가우시안 흐림과 유사한 매끄러운 흐림 효과가 나타납니다.

다음 예제에서는 Graphics 클래스의 `drawCircle()` 메서드를 사용하여 circle 객체를 만들고 여기에 흐림 필터를 적용합니다.

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// 윈을 그립니다 .
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// 표시 목록에 윈을 추가합니다 .
addChild(redDotCutout);
```

```
// 사각형에 흐림 필터를 적용합니다 .
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

그림자 필터

그림자는 대상 객체 위에 별도의 광원이 있다는 느낌을 줍니다. 이 광원의 위치 및 강도를 수정하여 여러 다양한 그림자 효과를 연출할 수 있습니다.

그림자 필터는 흐림 필터의 알고리즘과 유사한 알고리즘을 사용합니다. 그러나 가장 큰 차이점은 그림자 필터에서는 보다 많은 속성을 수정하여 알파, 색상, 오프셋 및 밝기 등의 다양한 광원 특성을 시뮬레이트할 수 있다는 점입니다.

그림자 필터를 사용하면 또한 그림자 스타일에 내부 또는 외부 그림자 및 녹아웃(또는 컷아웃) 모드 등의 사용자 정의 변형 옵션을 적용할 수 있습니다.

다음은 사각형 상자 Sprite를 만든 후 그림자 필터를 적용하는 코드입니다.

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// 상자를 그립니다 .
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// 상자에 그림자 필터를 적용합니다 .
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// 그림자 색상 , 흐림의 양 , 강도 , 품질은 물론 내부
// 그림자 효과와 녹아웃 효과 등의 기타 속성을 설정할
// 수도 있습니다 .

boxShadow.filters = [shadow];
```

광선 필터

GlowFilter 클래스는 표시 객체에 광선 효과를 적용하여 객체 아래쪽에서 빛이 비추어 객체가 부드럽게 빛나는 것처럼 보이도록 할 수 있습니다.

그림자 필터와 유사한 광선 필터에는 거리, 각도 및 광원의 색상을 수정할 수 있는 속성이 있어 다양한 효과 연출이 가능합니다. GlowFilter에는 또한 내부 또는 외부 광선 및 녹아웃 모드를 비롯한 광선 스타일을 수정할 수 있는 몇 가지 옵션이 있습니다.

다음은 Sprite 클래스를 사용하여 십자가를 만든 후 광선 필터를 적용하는 코드입니다.

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// 십자가 그래픽을 만듭니다 .
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// 십자가 모양에 광선 필터를 적용합니다 .
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

그래디언트 경사 필터

GradientBevelFilter 클래스를 사용하면 표시 객체 또는 BitmapData 객체에 강화된 경사 효과를 적용할 수 있습니다. 경사에 그래디언트 색상을 사용하면 경사의 공간 심도가 현저히 개선되기 때문에 가장자리에 보다 사실적인 3D 효과를 줄 수 있습니다.

다음 코드는 Shape 클래스의 `drawRect()` 메서드를 사용하여 rectangle 객체를 만들고 여기에 그래디언트 경사 필터를 적용합니다.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// 사각형을 그립니다 .
var box:Shape = new Shape();
```

```
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// 사각형에 그래디언트 경사를 적용합니다 .
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // 45 도의 반대 각도
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// 기타 속성을 사용하여 필터 강도와 내부 경사 및 녹아웃 효과에 대한 옵션을
// 설정할 수 있습니다 .

box.filters = [gradientBevel];

// 표시 목록에 그래픽을 추가합니다 .
addChild(box);
```

그래디언트 광선 필터

GradientGlowFilter 클래스를 사용하면 표시 객체 또는 BitmapData 객체에 강화된 광선 효과를 적용할 수 있습니다. 즉, 광선의 색상을 효율적으로 제어하여 한층 더 현실적인 광선 효과를 연출하도록 합니다. 또한 그래디언트 광선 필터를 사용하면 객체의 내부, 외부 또는 상단 가장자리에 그래디언트 광선을 적용할 수도 있습니다.

다음 예제에서는 Stage에 원을 그린 다음 그래디언트 광선 필터를 적용합니다. 마우스를 점점 오른쪽 아래로 옮길수록 가로 및 세로 방향 모두에서 흐림의 양이 증가합니다. 또한 Stage를 클릭할 때마다 흐림 강도가 높아집니다.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// 새 Shape 인스턴스를 만듭니다 .
var shape:Shape = new Shape();

// 모양을 그립니다 .
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
```

```
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// 스테이지에 모양을 배치합니다 .
addChild(shape);
shape.x = 100;
shape.y = 100;

// 그래디언트 광선을 정의합니다 .
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// 두 이벤트를 수신할 함수를 정의합니다 .
function onClick(event:MouseEvent):void
{
  gradientGlow.strength++;
  shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
  gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
  gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
  shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

예제: 기본 필터 결합

다음은 반복 액션을 만드는 Timer와 여러 기본 필터를 결합하여 신호등 애니메이션을 시뮬레이트하는 코드 예제입니다.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
```

```
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // 45 도의 반대 각도
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// 신호등의 사각형 배경을 그립니다 .
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// 신호등의 원을 3 개 그립니다 .
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// 표시 목록에 그래픽을 추가합니다 .
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// 신호등 사각형에 그래디언트 경사를 적용합니다 .
```

```
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance,
  angleInDegrees, colors, alphas, ratios, blurX, blurY, strength, quality,
  type, knockout);
box.filters = [gradientBevel];

// 내부 그림자 ( 꺼진 등에 사용 ) 와
// 광선 ( 켜진 등에 사용 ) 을 만듭니다 .
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3,
  3, 1, 1, true, false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false,
  false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1,
  false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false,
  false);

// 등의 시작 상태를 설정합니다 ( 파란불 켜짐 , 빨간불과 노란불 꺼짐 ).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// 카운트 값에 따라 필터를 교체합니다 .
function trafficControl(event:TimerEvent):void
{
  if (count == 4)
  {
    count = 1;
  }

  switch (count)
  {
    case 1:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [yellowGlow];
      goLight.filters = [innerShadow];
      break;
    case 2:
      stopLight.filters = [redGlow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [innerShadow];
      break;
    case 3:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [greenGlow];
      break;
  }

  count++;
}
```

```
// 타이머를 만들어 3초 간격으로 필터를 교체합니다.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

색상 매트릭스 필터

ColorMatrixFilter 클래스는 필터링된 객체의 색상 및 알파 값을 조작하는 데 사용됩니다. ColorMatrixFilter를 통해 채도 변경, 색조 회전(한 색상 범위에서 다른 색상 범위로 팔레트를 이동하는 것), 광도 변경 작업 등을 수행할 수 있으며 한 색상 채널의 값을 사용하고 이 값을 다른 채널에 적용하여 다른 색상 조작 효과를 연출할 수 있습니다.

개념적으로 이 필터는 소스 이미지의 픽셀을 하나씩 거쳐가면서 각 픽셀을 빨강, 녹색, 파랑 및 알파 요소로 분리합니다. 그런 다음 분리된 각 픽셀 값을 색상 매트릭스에 제공된 값으로 곱한 후 결과를 더해 화면에 표시될 해당 픽셀의 색상 값을 결정합니다. 필터의 `matrix` 속성은 최종 색상을 계산하는 데 사용되는 20개 숫자의 배열입니다. 색상 값 계산에 사용된 특정 알고리즘에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 ColorMatrixFilter 클래스의 `matrix` 속성을 설명하는 항목을 참조하십시오.

색상 매트릭스 필터에 대한 추가 정보 및 예제는 Adobe 개발자 센터 웹 사이트의 "Flash에서 변형, 색상 조정 및 회선 효과용 매트릭스 사용" 문서를 참조하십시오.

회선 필터

ConvolutionFilter 클래스를 사용하면 BitmapData 객체 또는 표시 객체에 흐리게 하기, 가장자리 감지, 선명하게 하기, 엠보싱 및 경사 등과 같은 다양한 범위의 이미지 변형을 적용할 수 있습니다.

회선 필터는 개념적으로 소스 이미지의 각 픽셀을 하나씩 거쳐가면서 각 픽셀 및 그 주변 픽셀의 값을 사용하여 해당 픽셀의 최종 색상을 결정합니다. 숫자 값 배열로 지정되는 행렬은 인접한 각 특정 픽셀 값이 최종 결과 값에 영향을 미치는 정도를 나타냅니다.

가장 일반적으로 사용되는 3x3 행렬을 예로 들어 보겠습니다. 아래 행렬에는 9개의 값이 있습니다.

```
N N N
N P N
N N N
```

Flash Player에서 특정 픽셀에 회선 필터를 적용하는 경우에는 픽셀 자체의 색상 값(이 예제에서는 "P")과 함께 주변 픽셀 값(이 예제에서 "N"으로 레이블 표시)을 검토합니다. 그러나 행렬의 값을 설정하여 각 픽셀이 결과 이미지에 영향을 미치는 우선 순위를 지정하십시오.

예를 들어 회선 필터를 사용하여 적용된 다음 행렬은 이미지를 원래 모습 그대로 유지합니다.

```
0 0 0
```

```
0  1  0
0  0  0
```

이미지가 변경되지 않는 것은 원래 픽셀 값에 최종 픽셀 색상을 결정하는 상대 강도가 1인 반면, 주변 픽셀 값의 상대 강도는 0으로 해당 색상이 최종 이미지에 반영되지 않기 때문입니다.

마찬가지로 다음 행렬은 이미지의 픽셀을 왼쪽으로 한 픽셀씩 이동하게 합니다.

```
0  0  0
0  0  1
0  0  0
```

이 경우 픽셀 자체는 최종 이미지의 해당 위치에 표시된 최종 픽셀 값에 영향을 미치지 않고, 오른쪽 픽셀 값만 해당 픽셀의 최종 값을 결정하는 데 사용됩니다.

ActionScript에서는 행렬의 행과 열 수를 지정하는 두 개의 속성과 값이 포함된 Array 인스턴스의 조합으로 행렬을 만들 수 있습니다. 다음 예제에서는 이미지를 로드하고 이미지 로딩이 끝나면 앞에서 설명한 행렬을 사용하여 해당 이미지에 회선 필터를 적용합니다.

```
// 이미지를 스테이지로 로드합니다 .
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
  // 회선 매트릭스를 만듭니다 .
  var matrix:Array = [ 0, 0, 0,
                       0, 0, 1,
                       0, 0, 0 ];

  var convolution:ConvolutionFilter = new ConvolutionFilter();
  convolution.matrixX = 3;
  convolution.matrixY = 3;
  convolution.matrix = matrix;
  convolution.divisor = 1;

  loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

이 코드에서는 행렬에 1 또는 0 이외의 다른 값을 사용했기 때문에 명확하지 않은 부분이 있습니다. 예를 들어 오른쪽에 1이 아닌 8이 있는 동일한 행렬은 동일한 액션을 수행합니다(왼쪽으로 픽셀 이동). 그러나 이미지의 색상에 영향을 미쳐 해당 이미지를 8배 밝게 만듭니다. 이는 최종 픽셀 색상이 해당 행렬 값에 원본 픽셀 값을 곱하고 이 두 값을 더한 다음 필터의 `divisor` 속성 값으로 나누어서 계산되기 때문입니다. 예제 코드에서 `divisor` 속성은 1로 설정됩니다. 일반적으로 원본 이미지와 동일한 색상 밝기를 유지하려는 경우 제수를 행렬 값의 합과 동일하게 해야 합니다. 따라서 행렬 값 합계가 8이고 제수가 1인 행렬의 결과 이미지는 원래 이미지보다 8배 가량 밝게 됩니다.

이 행렬의 효과는 그다지 두드러지지 않지만 다른 행렬 값을 사용하여 다양한 효과를 연출할 수 있습니다. 다음은 3x3 행렬을 통해 연출할 수 있는 여러 다양한 효과의 표준이 되는 몇 가지 행렬 값 집합입니다.

- 기본 흐림(제수 5)

```
0 1 0
1 1 1
0 1 0
```

- 선명 효과(제수 1)

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- 가장자리 감지(제수 1)

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- 엠보싱 효과(제수 1)

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

이러한 대부분의 효과에서 제수는 1입니다. 이는 양의 행렬 값에 음의 행렬 값을 추가하면 1(가장자리 감지의 경우 0이지만 `divisor` 속성 값은 0이 될 수 없음)이 되기 때문입니다.

위치 변경 맵 필터

DisplacementMapFilter 클래스는 BitmapData 객체(위치 변경 맵 이미지)의 픽셀 값을 사용하여 새 객체에서 위치 변경 효과를 수행합니다. 위치 변경 맵 이미지는 필터가 적용되는 실제 표시 객체나 BitmapData 인스턴스와는 일반적으로 다릅니다. 위치 변경 효과는 필터링된 이미지에 있는 픽셀의 위치를 변경하는 것 즉, 이미지의 픽셀을 원래 위치에서 다른 곳으로 이동하는 것입니다. 이 필터는 이동, 비틀기 또는 얼룩 효과를 연출하는 데 사용할 수 있습니다.

지정된 픽셀에 적용되는 위치 변경의 위치와 정도는 위치 변경 맵 이미지의 색상 값에 의해 결정됩니다. 필터를 사용하여 작업할 때는 맵 이미지를 지정하는 것 이외에도 다음 값을 지정하여 맵 이미지에서 위치 변경이 계산되는 방법을 제어합니다.

- 맵 포인트: 필터링된 이미지의 위치를 나타내며, 위치 변경 필터의 왼쪽 위 모서리가 적용됩니다. 이미지의 일부에만 필터를 적용하려면 이 값을 사용합니다.
- X 구성 요소: 맵 이미지의 색상 채널이 픽셀의 x 위치에 영향을 미칩니다.
- Y 구성 요소: 맵 이미지의 색상 채널이 픽셀의 y 위치에 영향을 미칩니다.
- X 배율: x축 위치 변경의 정도를 지정하는 승수 값입니다.
- Y 배율: y축 위치 변경의 정도를 지정하는 승수 값입니다.
- 필터 모드: 픽셀 이동으로 인해 생긴 빈 공간이 Flash Player에서 처리되는 방식을 결정합니다. DisplacementMapFilterMode 클래스에서 상수로 정의된 옵션에는 원본 픽셀 표시(필터 모드 `IGNORE`), 이미지의 다른 쪽에서 해당 픽셀 감싸기(필터 모드 `WRAP` - 기본값), 가장 가깝게 이동된 픽셀 사용(필터 모드 `CLAMP`) 또는 색상으로 공간 채우기(필터 모드 `COLOR`)가 있습니다.

기본적인 예제를 통해 위치 변경 맵 필터 작동 방식의 기본 개념을 알아보겠습니다. 다음 코드는 이미지를 로드한 후 해당 이미지를 Stage의 중앙에 배치하고 위치 변경 맵 필터를 적용하여 전체 이미지의 모든 픽셀을 왼쪽 가로 방향으로 이동시킵니다.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DdisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// 이미지를 스테이지로 로드합니다 .
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;
```

```
// 이미지 로드가 끝나면 이 함수가 호출됩니다 .
function setupStage(event:Event):void
{
  // 스테이지 중심에 로드된 이미지를 배치합니다 .
  loader.x = (stage.stageWidth - loader.width) / 2;
  loader.y = (stage.stageHeight - loader.height) / 2;

  // 위치 변경 맵 이미지를 만듭니다 .
  mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

  // 위치 변경 필터를 만듭니다 .
  displacementMap = new DisplacementMapFilter();
  displacementMap.mapBitmap = mapImage;
  displacementMap.mapPoint = new Point(0, 0);
  displacementMap.componentX = BitmapDataChannel.RED;
  displacementMap.scaleX = 250;
  loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

위치 변경 정의에 사용된 속성은 다음과 같습니다.

- 맵 비트맵: 위치 변경 비트맵은 코드에 의해 만들어진 새로운 BitmapData 인스턴스로서, 이 비트맵의 크기는 로드된 이미지의 크기와 일치하므로 위치 변경이 전체 이미지에 적용됩니다. 이 비트맵은 단색 빨강 픽셀로 채워집니다.
- 맵 포인트: 위치 변경이 전체 이미지에 적용되도록 값이 0, 0 점으로 설정됩니다.
- X 구성 요소: 이 값은 상수 `BitmapDataChannel.RED`로 설정되어 있으며 이는 맵 비트맵의 빨강 값이 x축을 따라 위치 변경(이동)된 픽셀의 양을 결정한다는 것을 의미합니다.
- X 배율: 이 값은 250으로 설정됩니다. 위치 변경(완전히 빨강인 맵 이미지에서 시작)의 전체 양은 이미지의 위치를 조금밖에(대략 1/2픽셀) 변경하지 못하므로 이 값을 1로 설정할 경우 이미지가 가로로 0.5픽셀만큼만 이동합니다. X 배율을 250으로 설정하면 이미지가 약 125픽셀만큼 이동합니다.

이러한 설정은 필터링된 이미지의 픽셀이 왼쪽으로 250픽셀만큼 이동하도록 합니다. 이동 방향(왼쪽 또는 오른쪽) 및 이동 정도는 맵 이미지에 있는 픽셀의 색상 값을 기반으로 합니다. 개념적으로 Flash Player는 필터링된 이미지의 픽셀을 하나씩 거쳐가면서(최소한 필터가 적용된 영역에 있는 픽셀, 이 예제의 경우에는 모든 픽셀) 각 픽셀에 대해 다음 작업을 수행합니다.

1. 맵 이미지에서 대응하는 픽셀을 찾습니다. 예를 들어 Flash Player에서는 필터링된 이미지의 왼쪽 위 모서리에 있는 픽셀의 위치 변경 정도를 계산할 때 맵 이미지의 왼쪽 위 모서리에 있는 픽셀을 확인합니다.
2. 맵 픽셀에서 특정 색상 채널의 값을 확인합니다. 이 예제에서 x 구성 요소 색상 채널이 빨강 채널이므로 Flash Player에서는 맵 이미지의 해당 픽셀에서 빨강 채널의 값을 확인합니다. 맵 이미지가 단색 빨강이므로 픽셀의 빨강 채널 값은 0xFF 또는 255이며, 이 값이 위치 변경 값으로 사용됩니다.

3. 위치 변경 값을 "중간" 값(0과 255의 중간 값인 127)과 비교합니다. 위치 변경 값이 중간 값보다 낮으면 픽셀이 양의 방향(x축 위치 변경의 경우 오른쪽, y축 위치 변경의 경우 아래쪽)으로 이동합니다. 반대로 이 예제에서처럼 위치 변경 값이 중간 값보다 높으면 픽셀이 음의 방향(x축 위치 변경의 경우 왼쪽, y축 위치 변경의 경우 위쪽)으로 이동합니다. 보다 정확히 설명하자면 Flash Player는 127에서 위치 변경 값을 감산하며, 그 결과(양의 결과 또는 음의 결과)가 적용할 위치 변경의 상대적 양이 됩니다.
4. 마지막으로 상대적인 위치 변경 값이 전체 위치 변경에서 차지하는 백분율을 파악하여 실제 위치 변경 양을 결정합니다. 이 예제에서 전체 빨강은 100%의 위치 변경을 의미합니다. 이 백분율을 이후 x 배율 또는 y 배율 값으로 곱하면 적용할 위치 변경의 픽셀 수가 산출됩니다. 이 예제에서 100%에 승수 250을 곱하면 위치 변경의 양이 구해집니다(대략 왼쪽으로 125픽셀).

y 구성 요소 및 y 배율에 지정된 값이 없기 때문에 기본값(위치 변경이 수행되지 않음)이 사용되었으며 결과 이미지가 세로 방향으로 이동하지 않습니다.

기본 필터 모드 설정인 `WRAP`이 이 예제에 사용되어 픽셀을 왼쪽으로 이동하면 오른쪽의 빈 공간이 이미지의 왼쪽 가장자리로 이동된 픽셀로 채워집니다. 이 값을 활용하여 다른 효과를 확인해볼 수 있습니다. 예를 들어, 위치 변경 속성이 설정된 코드 부분(`loader.filters = [displacementMap]` 행 앞)에 다음 행을 추가하면 이미지가 스테이지 전체에 퍼진 것처럼 보입니다.

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

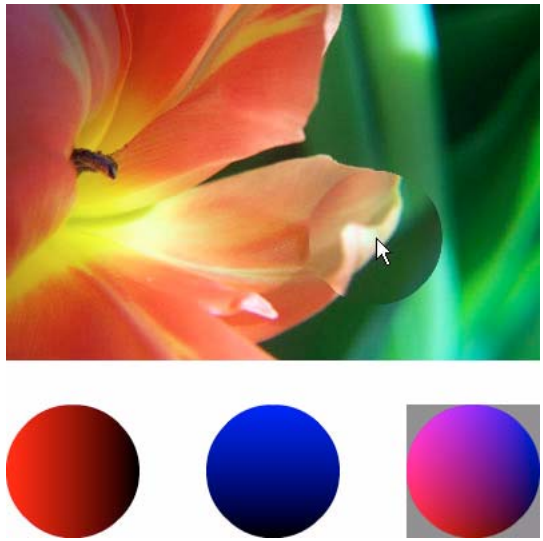
보다 복잡한 예제의 경우 다음 샘플에서 위치 변경 맵 필터를 사용하여 이미지에 돋보기 효과를 만듭니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// 위치 변경 맵 이미지를 구성할 그래디언트 원을 만듭니다 .
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [ 0xFF0000, 0x000000 ];
var blueColors:Array = [ 0x0000FF, 0x000000 ];
var alphas:Array = [ 1, 1 ];
var ratios:Array = [ 0, 255 ];
```

```
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios,
  xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios,
  yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// 화면 아래쪽에 윈을 참조용으로 배치합니다 .
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// 이미지를 스테이지로 로드합니다 .
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

// 두 개의 그래디언트 윈을 결합하여 맵 이미지를 만듭니다 .
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false,
  0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false,
  0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
  BitmapDataChannel.BLUE);
yMap.dispose();

// 스테이지에 맵 이미지를 참조용으로 표시합니다 .
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// 이 함수는 마우스 위치에 위치 변경 맵 필터를 만듭니다 .
function magnify():void
```

```
{
  // 필터를 배치합니다.
  var filterX:Number = (loader.mouseX) - (map.width / 2);
  var filterY:Number = (loader.mouseY) - (map.height / 2);
  var pt:Point = new Point(filterX, filterY);
  var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
  xyFilter.mapBitmap = map;
  xyFilter.mapPoint = pt;
  // 맵 이미지의 빨강은 x 위치 변경을 제어합니다.
  xyFilter.componentX = BitmapDataChannel.RED;
  // 맵 이미지의 파랑은 y 위치 변경을 제어합니다.
  xyFilter.componentY = BitmapDataChannel.BLUE;
  xyFilter.scaleX = 35;
  xyFilter.scaleY = 35;
  xyFilter.mode = DisplacementMapFilterMode.IGNORE;
  loader.filters = [xyFilter];
}

// 마우스를 움직이면 이 함수가 호출됩니다.
// 마우스가 로드된 이미지 위에 있을 경우 필터가 적용됩니다.
function moveMagnifier(event:MouseEvent):void
{
  if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
  {
    magnify();
  }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

먼저 코드는 두 개의 그래디언트 원을 생성하고 해당 원은 서로 결합되어 위치 변경 맵 이미지를 형성합니다. 빨강 원은 x축 위치 변경(`xyFilter.componentX = BitmapDataChannel.RED`)을 만들고, 파랑 원은 y축 위치 변경(`xyFilter.componentY = BitmapDataChannel.BLUE`)을 만듭니다. 위치 변경 맵 이미지 모양에 대한 이해를 돕기 위해 코드에서 원본 원과 함께 해당 맵 이미지인 결합 원을 화면의 맨 아래에 추가합니다.



그런 다음 이미지를 로드하고 마우스 움직임에 따라 마우스 아래에 있는 이미지의 부분에 위치 변경 필터를 적용합니다. 위치 변경 맵 이미지로 사용된 그래디언트 원은 위치 변경된 영역이 마우스 포인터에서부터 퍼지도록 합니다. 위치 변경 맵 이미지의 회색 영역에서는 위치 변경이 발생하지 않습니다. 회색 색상은 `0x7F7F7F`입니다. 회색 음영의 파랑 채널 및 빨강 채널은 해당 색상 채널의 중간 음영과 정확히 일치하므로 맵 이미지의 회색 영역에서는 위치 변경이 발생하지 않습니다. 마찬가지로 원 중심에서도 위치 변경이 발생하지 않습니다. 색상에 회색이 없다고 하더라도 파랑 채널과 빨강 채널이 중간 회색의 파랑 채널 및 빨강 채널과 똑같고, 위치 변경을 발생시키는 색상이 파랑 및 빨강이기 때문에 원 중심에서 위치 변경이 발생하지 않습니다.

예제: 필터 워크벤치

필터 워크벤치에서 제공하는 간단한 사용자 인터페이스를 통해 이미지에 다양한 필터를 적용해 보고 ActionScript에서 동일한 효과를 내는 결과 코드를 확인할 수 있습니다. 위 예제 및 해당 소스 코드 다운로드 방법에 대한 설명은 www.adobe.com/go/learn_fl_filters_kr을 참조하십시오.

제 16 장

무비 클립을 사용한 작업

16

MovieClip 클래스는 Adobe Flash CS3 Professional에서 생성한 애니메이션 및 무비 클립 심볼의 기본 클래스입니다. MovieClip 클래스는 표시 객체의 모든 비헤이비어 및 기능은 물론 무비 클립의 타임라인을 제어하기 위한 추가 속성 및 메서드까지 가지고 있습니다. 이 장에서는 ActionScript를 사용하여 무비 클립 재생을 제어하고 동적으로 무비 클립을 만드는 방법에 대해 설명합니다.

목차

무비 클립의 기초

무비 클립을 사용한 작업 소개

무비 클립은 Flash 제작 도구를 사용하여 애니메이션 내용을 만들고 ActionScript를 사용하여 그 내용을 제어하는 사람들에게 중요한 요소입니다. Flash에서 무비 클립 심볼을 만들 때마다 해당 Flash 문서의 라이브러리에 심볼이 추가됩니다. 기본적으로 이 심볼은 MovieClip 클래스의 인스턴스가 되고 MovieClip 클래스의 속성과 메서드를 가지게 됩니다.

무비 클립 심볼의 인스턴스를 스테이지에 배치하면 ActionScript를 사용하여 무비 클립의 재생을 변경하지 않는 한, 무비 클립이 자동으로 타임라인을 따라 진행됩니다(프레임이 둘 이상일 경우). MovieClip 클래스를 구별하는 것은 이 타임라인으로, Flash 제작 도구에서 모션 트윈이나 모양 트윈을 통해 애니메이션을 만들 수 있습니다. 반면 Sprite 클래스의 인스턴스인 표시 객체의 경우에는 프로그래밍을 통해 값을 변경하는 방법으로만 애니메이션을 만들 수 있습니다.

이전 버전의 ActionScript에서는 MovieClip 클래스가 Stage의 모든 인스턴스의 기본 클래스였습니다. ActionScript 3.0에서 무비 클립은 스크린에 표시할 수 있는 여러 표시 객체 중 하나일 뿐입니다. 표시 객체의 기능에 타임라인이 필요하지 않은 경우 MovieClip 클래스 대신 Shape 클래스나 Sprite 클래스를 사용하면 렌더링 성능을 높일 수 있습니다. 작업에 적합한 표시 객체를 선택하는 것에 대한 자세한 내용은 370페이지의 “DisplayObject 하위 클래스 선택”을 참조하십시오.

일반적인 무비 클립 작업

이 장에서는 다음과 같은 일반적인 무비 클립 관련 작업에 대해 설명합니다.

- 무비 클립 재생 및 중단
- 반대 방향으로 무비 클립 재생
- 무비 클립 타임라인의 특정 시점으로 재생 헤드 이동
- ActionScript에서 프레임 레이블 사용
- ActionScript에서 장면 정보 액세스
- ActionScript를 사용하여 라이브러리 무비 클립 심볼 인스턴스 만들기
- 이전 Flash Player 버전용으로 만든 파일을 비롯하여 외부 SWF 파일 로드 및 제어
- 런타임 시 로드해서 사용할 그래픽 에셋을 생성하기 위한 ActionScript 시스템 구축

중요한 개념 및 용어

다음은 이 장에서 사용된 중요 용어 참조 목록입니다.

- AVM1 SWF: ActionScript 1.0 또는 ActionScript 2.0으로 만든 SWF 파일로 대개 Flash Player 8 또는 이전 버전을 대상으로 합니다.
- AVM2 SWF: ActionScript 3.0을 사용하여 생성하는 Adobe Flash Player 9용 SWF 파일입니다.
- 외부 SWF: 프로젝트 SWF 파일과는 별개로 생성된 SWF 파일로, 프로젝트 SWF 파일로 로드되어 이 프로젝트 SWF 파일 내에서 재생됩니다.
- 프레임: 타임라인의 가장 작은 시간 구획입니다. 영화 필름스트립과 같이 각 프레임은 특정 시점의 애니메이션 스냅샷과 유사하며, 이러한 프레임을 차례로 빠르게 재생하면 애니메이션 효과가 연출됩니다.
- 타임라인: 무비 클립의 애니메이션 시퀀스를 구성하는 일련의 프레임을 은유적으로 표현한 것입니다. MovieClip 객체의 타임라인은 Flash 제작 도구의 타임라인에 해당합니다.
- 재생 헤드: 특정 순간에 표시되는 타임라인 내 위치(프레임)를 나타내는 표시자입니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장은 ActionScript의 무비 클립 작업을 다루므로, 기본적으로 이 장의 모든 코드 샘플은 생성되어 스테이지에 배치되어 있는 무비 클립 심볼을 조작하는 목적으로 작성되었습니다. 샘플을 테스트하려면 코드로 심볼을 조작한 결과를 Flash Player에서 확인해야 합니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. 스테이지에 무비 클립 심볼 인스턴스를 만듭니다. 예를 들어, 모양을 하나 그려 그 모양을 선택한 다음 [수정] > [심볼로 변환]을 선택한 후, 심볼에 이름을 지정합니다.
5. 무비 클립을 선택한 상태에서, 속성 관리자에서 무비 클립에 인스턴스 이름을 지정합니다. 이 이름은 예제 코드 샘플에서 해당 무비 클립에 대해 사용한 이름과 일치해야 합니다. 예를 들어, 코드 샘플에서 `myMovieClip`이라는 무비 클립을 조작하려는 경우 무비 클립 인스턴스 이름도 `myMovieClip`으로 지정해야 합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   코드 샘플에 지정된 대로 무비 클립을 코드 조작한 결과가 스크린에 표시됩니다.

예제 코드 목록 테스트와 관련한 기타 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

MovieClip 객체를 사용한 작업

SWF 파일을 제작하면 스테이지의 모든 무비 클립 심볼 인스턴스가 MovieClip 객체로 변환됩니다. 속성 관리자의 [인스턴스 이름] 필드에 무비 클립 심볼의 인스턴스 이름을 지정하면 ActionScript에 심볼을 사용할 수 있습니다. SWF 파일이 만들어지면 스테이지에 MovieClip 인스턴스를 만드는 코드가 생성되고 인스턴스 이름을 사용하여 변수가 선언됩니다. 이름이 지정된 무비 클립 내에 이름이 지정된 다른 무비 클립이 중첩되어 있는 경우 그러한 자식 무비 클립은 부모 무비 클립의 속성으로 처리되므로 도트 구문을 사용하여 자식 무비 클립에 액세스할 수 있습니다. 예를 들어, 인스턴스 이름이 `childClip`인 무비 클립이 인스턴스 이름이 `parentClip`인 다른 클립 내에 중첩되어 있는 경우, 다음 코드를 호출하여 자식 클립의 타임라인 애니메이션이 재생되도록 할 수 있습니다.

```
parentClip.childClip.play()
```

ActionScript 2.0 MovieClip 클래스의 이전 메서드와 속성 중 일부는 그대로이지만 변경된 것도 있습니다. 밑줄로 시작하는 속성 이름은 모두 변경되었습니다. 예를 들어, `_width` 및 `_height` 속성은 이제 `width` 및 `height`로 액세스되고 `_xscale` 및 `_yscale`은 이제 `scaleX` 및 `scaleY`로 액세스됩니다. MovieClip 클래스의 속성 및 메서드에 대한 전체 목록은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*를 참조하십시오.

무비 클립 재생 제어

Flash에서는 타임라인 메타포를 사용하여 애니메이션이나 상태 변경을 나타냅니다. 타임라인을 사용하는 시각적 요소는 MovieClip 객체이거나 MovieClip 클래스에서 확장해야 합니다. ActionScript 명령으로 무비 클립을 중지하거나 재생하거나 타임라인의 다른 지점으로 이동할 수는 있지만 ActionScript를 사용하여 동적으로 타임라인을 만들거나 특정 프레임에 내용을 추가할 수는 없습니다. 이러한 작업은 Flash 제작 도구에서만 가능합니다.

MovieClip을 재생하는 경우 SWF 파일의 프레임 속도에 의해 지정된 속도로 타임라인을 진행합니다. 또는 ActionScript에서 `Stage.frameRate` 속성을 설정하여 이 설정을 덮어쓸 수 있습니다.

무비 클립 재생 및 재생 중지

`play()` 및 `stop()` 메서드를 사용하면 타임라인 전체에서 무비 클립의 기본 컨트롤을 사용할 수 있습니다. 예를 들어, 자전거가 스크린을 가로질러 움직이는 애니메이션이 포함된 무비 클립 심볼이 스테이지에 있고 이 클립의 인스턴스 이름이 `bicycle`로 지정되었다고 가정해 봅니다. 이 경우 기본 타임라인의 키프레임에 다음 코드가 첨부되면,

```
bicycle.stop();
```

자전거는 움직이지 않게 됩니다. 다시 말해 애니메이션이 재생되지 않습니다. 자전거의 움직임은 다른 사용자 조작을 통해서도 시작할 수 있습니다. 예를 들어, `startButton`이라는 버튼이 있는 경우, 기본 타임라인의 키프레임에 다음 코드를 적용하면 버튼을 클릭해야 애니메이션이 재생되도록 만듭니다.

```
// 버튼을 클릭하면 이 함수가 호출됩니다 .
// 재생할 자전거 애니메이션입니다 .
function playAnimation(event:MouseEvent):void
{
  bicycle.play();
}
// 버튼에 함수를 리스너로 등록합니다 .
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

빨리 감기 및 되감기

play() 및 stop() 메서드를 사용하는 방법 외에도 무비 클립에서 재생을 제어하는 다른 방법이 있습니다. nextFrame() 및 prevFrame() 메서드를 사용하여 재생 헤드를 타임라인을 따라 수동으로 빨리 감거나 되감을 수도 있습니다. 이러한 메서드 중 하나를 호출하면 재생이 중지되고 재생 헤드가 앞으로 또는 뒤로 각각 이동합니다.

play() 메서드를 사용하는 것은 무비 클립 객체의 enterFrame 이벤트가 트리거될 때마다 nextFrame()을 호출하는 것과 비슷합니다. 이러한 맥락에서, 다음과 같이 enterFrame 이벤트에 대한 이벤트 리스너를 만들고 bicycle이 해당 리스너 함수 내의 이전 프레임으로 이동하도록 지시하는 방법으로 bicycle 무비 클립의 되감기를 실행할 수 있습니다.

```
// enterFrame 이벤트를 트리거하면 이 함수가 호출됩니다 .
// 즉 , 프레임마다 한 번씩 함수가 호출됩니다 .
function everyFrame(event:Event):void
{
  if (bicycle.currentFrame == 1)
  {
    bicycle.gotoAndStop(bicycle.totalFrames);
  }
  else
  {
    bicycle.prevFrame();
  }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

일반적으로 재생할 때 무비 클립에 둘 이상의 프레임이 있는 경우 이 무비 클립을 재생하면 무한정 반복됩니다. 즉, 마지막 프레임을 지나면 프레임 1로 돌아옵니다. prevFrame() 또는 nextFrame()을 사용하면 이러한 동작이 자동으로 발생하지 않습니다. 다시 말해서, 재생 헤드가 프레임 1에 있을 때 prevFrame()을 호출해도 재생 헤드가 마지막 프레임으로 이동하지 않습니다. 위 예제에서 if 조건은 재생 헤드가 첫 번째 프레임으로 되감기를 진행했는지 확인하고, 재생 헤드를 마지막 프레임 앞에 오도록 설정하여 되감기를 실행하는 무비 클립이 효과적으로 반복되도록 만듭니다.

다른 프레임으로 이동 및 프레임 레이블 사용

새 프레임에 무비 클립을 보내는 작업은 간단합니다. gotoAndPlay() 또는 gotoAndStop()을 호출하면 무비 클립이 매개 변수로 지정된 프레임 번호로 이동합니다. 또는 프레임 레이블 이름과 일치하는 문자열을 전달할 수 있습니다. 타임라인의 모든 프레임에 레이블을 지정할 수 있습니다. 그렇게 하려면 타임라인에서 프레임을 선택하고 속성 관리자의 [프레임 레이블] 필드에 이름을 입력합니다.

특히 복잡한 무비 클립을 만드는 경우에 번호 대신 프레임 레이블을 사용하는 것이 좋습니다. 애니메이션의 프레임, 레이어 및 트윈 개수가 많아지면 중요한 프레임에 대해 무비 클립 비헤이비어의 변화를 나타내는 설명(예: “off”, “walking” 또는 “running”)으로 레이블을 지정하는 것이 좋습니다. 레이블이 지정된 프레임으로 가는 ActionScript 호출은 특정 프레임 번호가 아닌 하나의 참조, 즉 레이블을 가리키므로 이렇게 하면 코드의 가독성도 향상되고 유연성도 증가합니다. 나중에 애니메이션의 특정 선분을 다른 프레임으로 이동할 경우 새 위치에서 해당 프레임의 레이블을 동일하게 유지하는 한 ActionScript 코드를 변경할 필요가 없습니다.

ActionScript 3.0은 프레임 레이블을 코드로 표시할 수 있도록 FrameLabel 클래스를 제공합니다. 이 클래스의 각 인스턴스는 단일 프레임 레이블을 나타내며, `name` 속성(속성 관리자에 지정된 프레임 레이블 이름을 나타냄) 및 `frame` 속성(레이블이 배치될 타임라인 프레임의 프레임 번호를 나타냄)을 가지고 있습니다.

무비 클립 인스턴스와 연관된 FrameLabel 인스턴스에 액세스할 수 있도록 MovieClip 클래스에는 FrameLabel 객체를 직접 반환하는 두 개의 속성이 포함되어 있습니다. `currentLabels` 속성은 무비 클립 전체 타임라인의 모든 FrameLabel 객체로 이루어진 배열을 반환합니다. `currentLabel` 속성은 최근에 타임라인에 나타난 FrameLabel 객체 하나를 반환합니다.

로봇이라는 무비 클립을 만들었으며 애니메이션의 여러 가지 상태에 레이블을 지정했다고 가정할 경우 다음 코드와 같이 로봇의 현재 상태에 액세스할 수 있는 `currentLabel` 속성을 확인하는 조건을 설정할 수 있습니다.

```
if (robot.currentLabel.name == "walking"
{
  // 작업을 수행하십시오 .
}
```

장면을 사용한 작업

Flash 제작 환경에서는 SWF 파일이 진행되는 타임라인을 연속으로 보여 주는 장면을 사용할 수 있습니다. `gotoAndPlay()` 또는 `gotoAndStop()` 메서드의 두 번째 매개 변수를 사용하면 재생 헤드를 보낼 수 있는 장면을 지정할 수 있습니다. 모든 FLA 파일은 첫 장면으로만 시작하지만 새 장면을 만들 수도 있습니다.

장면에는 단점이 많으므로 장면을 사용하는 것이 항상 최선은 아닙니다. 여러 장면이 포함된 Flash 문서는 유지 관리가 어려울 수 있습니다. 특히 제작자가 여러 명인 경우 더욱 그렇습니다. 제작 프로세스 동안 모든 장면을 하나의 타임라인에 병합하므로 장면이 여러 개인 경우 대역폭이 불충분할 수도 있습니다. 따라서 장면이 재생된 적이 없는 경우에도 모든 장면이 점진적으로 다운로드될 수 있습니다. 따라서 여러 타임라인을 기반으로 하는 장편 애니메이션을 구성하는 경우를 제외하고는 여러 장면을 사용하지 않는 것이 좋습니다.

MovieClip 클래스의 `scenes` 속성은 SWF 파일에 있는 모든 장면을 나타내는 Scene 객체의 배열을 반환합니다. `currentScene` 속성은 현재 재생 중인 장면을 나타내는 Scene 객체를 반환합니다.

Scene 클래스에는 장면 정보를 제공하는 여러 속성이 있습니다. `labels` 속성은 해당 장면 내의 프레임 레이블을 나타내는 FrameLabel 객체의 배열을 반환합니다. `name` 속성은 장면의 이름을 문자열로 반환합니다. `numFrames` 속성은 장면의 전체 프레임 수를 나타내는 int를 반환합니다.

ActionScript를 사용하여 MovieClip 객체 만들기

Flash에서 스크린에 내용을 추가하는 한 가지 방법은 라이브러리의 에셋을 스테이지로 드래그하는 것입니다. 그러나 이외에 다른 작업 과정도 있습니다. 복잡한 프로젝트의 경우 일반적으로 숙련된 개발자는 프로그래밍 방식으로 무비 클립을 만드는 경우가 많습니다. 이러한 방식은 코드를 다시 사용하기가 쉽고, 컴파일 시간 속도가 빠르며, ActionScript에서만 가능한 정교한 수정 작업을 수행할 수 있다는 장점이 있습니다.

ActionScript 3.0의 표시 목록 API에서는 동적으로 MovieClip 객체를 만드는 프로세스가 간편해졌습니다. MovieClip 인스턴스를 표시 목록에 추가하지 않고도 직접 인스턴스화할 수 있으므로 제어 기능을 그대로 유지하면서 작업을 유연하고 간편하게 수행할 수 있습니다.

ActionScript 3.0에서 프로그래밍 방식으로 무비 클립(또는 기타 표시 객체) 인스턴스를 만들면, 표시 객체 컨테이너에서 `addChild()` 또는 `addChildAt()` 메서드를 호출하여 만든 인스턴스를 표시 목록에 추가해야만 스크린에 나타납니다. 따라서 무비 클립을 만들어 속성을 설정하고, 메서드를 스크린에 렌더링하기 전에 호출할 수 있습니다. 표시 목록을 사용한 작업에 대한 자세한 내용은 360페이지의 “표시 객체 컨테이너 작업”을 참조하십시오.

ActionScript의 라이브러리 심볼 내보내기

기본적으로 Flash 문서의 라이브러리에 있는 무비 클립 심볼 인스턴스는 동적으로 만들 수 없습니다. 다시 말해서, ActionScript를 이용해서만 만들 수 있습니다. 이는 심볼을 ActionScript에서 사용하기 위해 내보낼 때마다 SWF 파일의 크기가 늘어날 뿐 아니라 일부 심볼은 스테이지용에서 사용할 수 있도록 만들어지지 않았기 때문입니다. 이러한 이유로 ActionScript에서 심볼을 사용하려면 해당 심볼을 ActionScript용으로 내보내도록 지정해야만 합니다.

심볼을 ActionScript에 사용할 수 있도록 내보내려면:

1. [라이브러리] 패널에서 심볼을 선택하고 [심볼 속성] 대화 상자를 엽니다.
2. 필요한 경우 [고급 설정]을 활성화합니다.
3. [링크] 섹션에서 [ActionScript에 내보내기] 체크 상자를 선택합니다.
   그러면 [클래스] 및 [기본 클래스] 필드가 활성화됩니다.

기본적으로 [클래스] 필드에는 공백이 제거된 심볼 이름(예: 이름이 "Tree House"인 심볼의 경우 "TreeHouse")이 표시됩니다. 심볼이 비헤이비어에 사용자 정의 클래스를 사용하도록 지정하려면 이 필드에 패키지 이름을 포함한 전체 클래스 이름을 입력합니다. 비헤이비어를 추가하지 않고 ActionScript에 심볼의 인스턴스를 만들려는 경우 클래스 이름을 그대로 두면 됩니다.

[기본 클래스] 필드의 기본값은 `flash.display.MovieClip`입니다. 심볼이 기타 사용자 정의 클래스의 기능을 확장하도록 만들려면, 해당 클래스가 Sprite 또는 MovieClip 클래스를 확장하는 경우 클래스의 이름을 지정할 수 있습니다.

4. [확인] 버튼을 눌러 변경 내용을 저장합니다.

   이때 Flash에서 지정된 클래스가 정의되어 있는 외부 ActionScript 파일을 찾지 못하면(예를 들어, 심볼에 비헤이비어를 추가할 필요가 없는 경우) 다음과 같은 경고 메시지가 표시됩니다.

   이 클래스의 정의는 클래스 경로에서 찾을 수 없으므로 내보낼 때 SWF 파일에서 자동으로 생성됩니다.

   라이브러리 심볼에 MovieClip 클래스의 기능 외에 고유한 기능이 필요하지 않다면 이 경고를 무시해도 됩니다.

사용자가 심볼에 대한 클래스를 제공하지 않으면 Flash에서 다음과 같이 심볼에 대한 클래스를 만듭니다.

```
package
{
  import flash.display.MovieClip;

  public class ExampleMovieClip extends MovieClip
  {
    public function ExampleMovieClip()
    {
    }
  }
}
```

심볼에 기타 ActionScript 기능을 추가하려면 이 구조에 해당 속성과 메서드를 추가합니다. 예를 들어, 폭과 높이가 각각 50픽셀인 원이 포함된 무비 클립 심볼이 있고, 이름이 Circle인 클래스를 사용하여 심볼을 ActionScript에 내보내도록 지정했다고 가정해 봅시다. 다음 코드가 Circle.as 파일에 배치되면 MovieClip 클래스를 확장하고 추가 메서드 `getArea()` 및 `getCircumference()`가 있는 심볼을 제공합니다.

```
package
{
  import flash.display.MovieClip;

  public class Circle extends MovieClip
  {
    public function Circle()
```

```
    {
    }

    public function getArea():Number
    {
        // 수식은 Pi x R-제곱입니다 .
        return Math.PI * Math.pow((width / 2), 2);
    }

    public function getCircumference():Number
    {
        // 수식은 Pi x 지름입니다 .
        return Math.PI * width;
    }
  }
}
```

다음 코드가 Flash 문서 프레임 1의 키프레임에 배치되면 심볼의 인스턴스를 만들어 스크린에 표시합니다.

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

이 코드는 개별 에셋을 스테이지로 드래그하는 대안으로 ActionScript 기반 인스턴스화를 보여 줍니다. 무비 클립의 모든 속성과 Circle 클래스에서 정의한 사용자 정의 메서드가 있는 원이 만들어집니다. 이것은 매우 기본적인 예로서, 라이브러리 심볼은 클래스에서 원하는 수만큼 속성과 메서드를 지정할 수 있습니다.

ActionScript 기반 인스턴스화는 수동으로 정렬하기에는 지루한 대량의 인스턴스를 동적으로 만들 수 있으므로 강력합니다. 또한 각 인스턴스를 만들 때 해당 속성을 사용자 정의할 수 있으므로 유연합니다. 루프를 사용하여 여러 Circle 인스턴스를 동적으로 만들면 이러한 장점을 모두 이용할 수 있습니다. 앞서 설명한 Flash 문서 라이브러리의 Circle 심볼 및 클래스를 사용하여 프레임 1의 키프레임에 다음 코드를 배치합니다.

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
  // 새 Circle 인스턴스를 만듭니다 .
  var c:Circle = new Circle();
  // x 좌표에 새 Circle을 배치합니다 . 이 x 좌표를 기준으로 스테이지
  // 전체에 원이 고른 간격으로 배치됩니다 .
  c.x = (stage.stageWidth / totalCircles) * i;
  // 스테이지의 세로 중심에 Circle 인스턴스를 배치합니다 .
```

```
    c.y = stage.stageHeight / 2;
    // Circle 인스턴스를 임의의 색상으로 변경합니다 .
    c.transform.colorTransform = getRandomColor();
    // 현재 타임라인에 Circle 인스턴스를 추가합니다 .
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // 빨강 , 녹색 및 파랑 색상 채널에 대한 임의의 값을 생성합니다 .
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // 임의의 색상으로 ColorTransform 객체를 만들어서 반환합니다 .
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

이 예는 코드를 사용하여 심볼에 대한 여러 개의 인스턴스를 빨리 만들고 사용자 정의할 수 있는 방법을 보여 줍니다. 루프 내에 있는 현재 개수에 따라 각 인스턴스가 배치되고, `transform` 속성(Circle이 MovieClip 클래스를 확장하여 상속함)을 설정하여 각 인스턴스에 임의의 색이 지정됩니다.

외부 SWF파일 로드

ActionScript 3.0에서 SWF 파일은 Loader 클래스를 사용하여 로드됩니다. 외부 SWF 파일을 로드하려면 ActionScript에서는 다음 네 가지를 수행합니다.

1. 파일의 URL을 사용하여 새 URLRequest 객체를 만듭니다.
2. 새 Loader 객체를 만듭니다.
3. 이 Loader 객체의 `load()` 메서드를 호출하여 URLRequest 인스턴스를 매개 변수로 전달합니다.
4. 표시 객체 컨테이너(예: Flash 문서의 기본 타임라인)의 `addChild()` 메서드를 호출하여 Loader 인스턴스를 표시 목록에 추가합니다.

완성된 코드는 다음과 같습니다.

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

위와 동일한 코드를 사용하여 JPEG, GIF, PNG 이미지 등의 외부 이미지 파일을 로드할 수 있습니다. 이때 SWF 파일 URL이 아니라 해당 이미지 파일의 URL을 지정합니다. SWF 파일은 이미지 파일과 달리 ActionScript를 포함할 수 있습니다. SWF 파일 로드 프로세스와 이미지 로드가 동일하지만, ActionScript를 사용하여 외부 SWF 파일과 통신하려면 외부 SWF 파일 로드 시 로드하는 SWF 파일과 로드되는 SWF 파일이 모두 동일한 보안 샌드박스에 있어야 합니다. 또한 외부 SWF 파일의 클래스 네임스페이스가 로드하는 SWF 파일의 클래스 네임스페이스와 동일한 경우 네임스페이스의 충돌을 방지하기 위해 로드되는 SWF 파일에 대한 새 응용 프로그램 도메인을 만들어야 할 수도 있습니다. 보안 및 응용 프로그램 도메인에 대한 자세한 내용은 656페이지의 “ApplicationDomain 클래스 사용” 및 725페이지의 “SWF 파일 및 이미지 로드”를 참조하십시오.

외부 SWF 파일이 성공적으로 로드되면 `Loader.content` 속성을 통해 액세스할 수 있습니다. 외부 SWF 파일이 ActionScript 3.0용으로 제작된 경우 이 파일은 확장하는 클래스에 따라 무비 클립이나 스프라이트 중 하나가 됩니다.

이전 SWF 파일을 로드할 때 고려할 사항

이전 버전의 ActionScript를 사용하여 외부 SWF 파일을 제작한 경우 고려해야 할 중요한 제한 사항이 있습니다. AVM2(ActionScript Virtual Machine 2)에서 실행되는 ActionScript 3.0 SWF 파일과는 달리 ActionScript 1.0 또는 2.0용으로 제작된 SWF 파일은 AVM1(ActionScript Virtual Machine 1)에서 실행됩니다.

AVM1 SWF 파일이 로드되면 로드된 객체(`Loader.content` 속성)는 AVM1Movie 객체가 됩니다. AVM1Movie 인스턴스는 MovieClip 인스턴스와 동일하지 않습니다. 표시 객체이지만 무비 클립과 달리 타임라인 관련 메서드나 속성이 없습니다. 부모 파일인 AVM2 SWF는 로드된 AVM1Movie 객체의 속성, 메서드 또는 객체에 대한 액세스 권한이 없습니다.

AVM2 SWF 파일에 의해 로드된 AVM1 SWF 파일에는 추가 제한 사항이 있습니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 AVM1Movie 클래스 샘플을 참조하십시오.

예제: RuntimeAssetsExplorer

[ActionScript에 내보내기] 기능은 둘 이상의 프로젝트에 유용하게 사용할 수 있는 라이브러리의 경우 특히 장점이 많습니다. ActionScript로 내보낸 심볼은 해당 SWF 파일에서만이 아니라 이것을 로드하는 동일한 보안 샌드박스 내의 모든 SWF 파일에 사용할 수 있습니다. 이런 방식으로 하나의 Flash 문서는 그래픽 에셋을 보유하기 위한 용도로만 지정된 SWF 파일을 생성할 수 있습니다. 이 기술은 "래터" SWF 파일을 만든 다음 런타임 시 그래픽 에셋 SWF 파일을 로드하는 개발자와 디자이너가 함께 시각적 에셋을 사용하는 큰 프로젝트에 특히 유용합니다. 이 메서드를 사용하여 일련의 버전 관리된 파일을 유지 관리할 수 있습니다. 이러한 파일에서는 그래픽 에셋이 프로그래밍 개발 과정의 영향을 받지 않습니다.

RuntimeAssetsExplorer 응용 프로그램은 RuntimeAsset의 하위 클래스인 모든 SWF 파일을 로드하며 해당 SWF 파일의 사용 가능한 에셋을 찾아볼 수 있습니다. 다음은 이러한 경우에 대한 예제입니다.

- `Loader.load()`를 사용하여 외부 SWF 파일 로드
- ActionScript에 내보낸 라이브러리 심볼을 동적으로 만들기
- MovieClip 재생의 ActionScript 컨트롤

시작하기 전에 각 SWF 파일을 동일한 보안 샌드박스에 반드시 배치하십시오.
자세한 내용은 719페이지의 "보안 샌드박스"를 참조하십시오.

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. RuntimeAssetsExplorer 응용 프로그램 파일은 Samples/RuntimeAssetsExplorer 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
RuntimeAssetsExample.mxml 또는 RuntimeAssetsExample.fla	Flex(MXML) 또는 Flash(FLA)용 응용 프로그램의 사용자 인터페이스입니다.
GeometricAssets.as	RuntimeAsset 인터페이스를 구현하는 클래스에 대한 예제입니다.
GeometricAssets.fla	ActionScript용으로 내보내진 심볼을 포함한 GeometricAssets 클래스(FLA의 문서 클래스)에 연결된 FLA 파일입니다.
com/example/programmingas3/runtimeassetsexplorer/RuntimeLibrary.as	탐색기 컨테이너로 로드될 모든 런타임 에셋 SWF 파일을 반환할 필수 메서드를 정의하는 인터페이스입니다.

파일	설명
com/example/programmingas3/runtimeassetsexplorer/AnimatingBox.as	회전 상자 모양에 있는 라이브러리 심볼의 클래스입니다.
com/example/programmingas3/runtimeassetsexplorer/AnimatingStar.as	회전 별 모양에 있는 라이브러리 심볼의 클래스입니다.

런타임 라이브러리 인터페이스 구축

탐색기가 SWF 라이브러리와 적절히 상호 작용하려면 런타임 에셋 라이브러리의 구조를 정형화해야 합니다. 이러한 구조는 예상 구조를 보여 주는 메서드의 청사진이라는 점에서 클래스와 유사하지만 클래스와는 다르게 메서드 본문이 포함되지 않은 인터페이스를 만들어 정형화할 수 있습니다. 이 인터페이스는 런타임 라이브러리와 탐색기 모두에 서로 통신할 수 있는 방법을 제공합니다. 브라우저에 로드되는 런타임 에셋의 각 SWF는 이 인터페이스를 구현합니다. 인터페이스 및 그 활용에 대한 자세한 정보는 150페이지의 “인터페이스”를 참조하십시오.

RuntimeLibrary 인터페이스는 매우 간단합니다. 심볼을 내보내고 런타임 라이브러리에서 사용할 수 있는 클래스 경로 배열을 탐색기에 제공할 수 있는 함수만 있으면 됩니다. 이 작업을 수행할 수 있도록 인터페이스에 `getAssets()`라는 메서드 하나가 있습니다.

```
package com.example.programmingas3.runtimeassetexplorer
{
  public interface RuntimeLibrary
  {
    function getAssets():Array;
  }
}
```

에셋 라이브러리 SWF 파일 만들기

RuntimeLibrary 인터페이스를 정의하여 다른 SWF 파일로 로드할 수 있는 여러 에셋 라이브러리 SWF 파일을 만들 수 있습니다. 에셋의 개별 SWF 라이브러리를 만들려면 다음 네 가지 작업을 수행해야 합니다.

- 에셋 라이브러리 SWF 파일의 클래스 만들기
- 라이브러리에 포함된 개별 에셋의 클래스 만들기
- 실제 그래픽 에셋 만들기
- 그래픽 요소를 클래스와 연결하고 라이브러리 SWF 제작

RuntimeLibrary 인터페이스 구현을 위한 클래스 만들기

다음으로 RuntimeLibrary 인터페이스를 구현할 GeometricAssets 클래스를 만듭니다. 이 클래스가 FLA의 문서 클래스가 되며 이 클래스의 코드는 RuntimeLibrary 인터페이스와 매우 유사합니다. 단, 클래스 정의에는 `getAssets()` 메서드에 메서드 본문이 있다는 점이 다릅니다.

```
package
{
  import flash.display.Sprite;
  import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

  public class GeometricAssets extends Sprite implements RuntimeLibrary
  {
    public function GeometricAssets() {

    }
    public function getAssets():Array {
      return [
  "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",

  "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
    }
  }
}
```

두 번째 런타임 라이브러리를 만들려는 경우 자체 `getAssets()` 구현을 제공하는 다른 클래스(예: AnimationAssets)를 기초로 다른 FLA를 만들 수 있습니다.

각 MovieClip 에셋의 클래스 만들기

이 예제에서는 사용자 정의 에셋에 기능을 추가하지 않고 MovieClip 클래스만 확장할 것입니다. AnimatingStar에 대한 다음 코드는 AnimatingBox에 대한 코드와 비슷합니다.

```
package com.example.programmingas3.runtimeassetexplorer
{
  import flash.display.MovieClip;

  public class AnimatingStar extends MovieClip
  {
    public function AnimatingStar() {
    }
  }
}
```

라이브러리 제작

이제 새 FLA를 만들고 속성 관리자의 [문서 클래스] 필드에 GeometricAssets를 입력하여 MovieClip 기반 에셋을 새로운 클래스에 연결합니다. 이에 대한 예를 들기 위해 360개가 넘는 프레임에 하나의 시계 방향 회전을 만들기 위해 타임라인 트윈을 사용하는 매우 기본적인 모양 두 개를 만들 것입니다. `animatingBox` 및 `animatingStar` 심볼은 모두 [ActionScript에 내보내기]로 설정되고 [클래스] 필드는 `getAssets()` 구현에 지정된 각각의 클래스 경로로 설정됩니다. 표준 MovieClip 메서드를 하위 클래스로 사용하려고 하면 `flash.display.MovieClip`의 기본 클래스가 그대로 유지됩니다.

심볼의 내보내기 설정을 설정한 다음 FLA를 제작합니다. 이제 첫 번째 런타임 라이브러리가 만들어집니다. 이 SWF 파일은 다른 AVM2 SWF 파일에 로드될 수 있으며 새 SWF 파일에 대해 AnimatingBox 및 AnimatingStar 심볼을 사용할 수 있습니다.

다른 SWF 파일로 라이브러리 로드

마지막으로 설명할 기능은 에셋 탐색기의 사용자 인터페이스입니다. 이 예제에서 런타임 라이브러리 경로는 `ASSETS_PATH`라는 변수로 하드 코딩되어 있습니다. FileReference 클래스를 사용할 수도 있습니다. 예를 들어, 하드 드라이브에서 특정 SWF 파일을 탐색하는 인터페이스를 만들려는 경우에 FileReference 클래스를 사용합니다.

런타임 라이브러리가 성공적으로 로드되면 Flash Player는 `runtimeAssetsLoadComplete()` 메서드를 호출합니다.

```
private function runtimeAssetsLoadComplete(event:Event):void
{
  var rl:* = event.target.content;
  var assetList:Array = rl.getAssets();
  populateDropdown(assetList);
  stage.frameRate = 60;
}
```

이 메서드에서 변수 rl은 로드된 SWF 파일을 나타냅니다. 코드는 로드된 SWF 파일의 `getAssets()` 메서드를 호출하고 사용 가능한 에셋 목록을 가져온 다음, 이 에셋을 사용하여 `populateDropDown()` 메서드 호출로 사용 가능한 에셋 목록을 ComboBox 구성 요소에 채웁니다. 그리고 해당 메서드는 각 에셋의 전체 클래스 경로를 저장합니다. 사용자 인터페이스에서 [추가] 버튼을 클릭하면 `addAsset()` 메서드가 트리거됩니다.

```
private function addAsset():void
{
  var className:String = assetNameCbo.selectedItem.data;
  var AssetClass:Class = getDefinitionByName(className) as Class;
  var mc:MovieClip = new AssetClass();
  ...
}
```

이 메서드는 ComboBox에 현재 선택되어 있는 에셋의 클래스 경로를 가져오고 (`assetNameCbo.selectedItem.data`), flash.utils 패키지의 `getDefinitionByName()` 함수를 사용하여 에셋의 클래스에 대한 실제 참조를 가져와 에셋의 새 인스턴스를 만듭니다.

제 17 장

텍스트를 사용한 작업

17

ActionScript 3.0에서는 일반적으로 텍스트가 텍스트 필드 내에 표시되지만 경우에 따라 표시 목록의 속성(예: UI 구성 요소의 레이블)으로 표시될 수 있습니다. 이 장에서는 텍스트 필드에서 스크립트 정의 내용을 사용하는 방법과 사용자 입력을 통해 원격 파일에서 동적 텍스트나 Adobe Flash CS3 Professional에서 정의한 정적 텍스트를 사용하는 방법에 대해 설명합니다. ActionScript 3.0 프로그래머의 경우 텍스트 필드에 특정 내용을 구축하거나 텍스트의 소스를 지정한 다음 스타일과 서식을 사용하여 해당 텍스트의 모양을 설정할 수 있습니다. 또한 사용자가 텍스트를 입력하거나 하이퍼링크를 클릭하여 사용자 이벤트에 응답할 수 있습니다.

목차

텍스트를 사용한 작업의 기초

텍스트를 사용한 작업 소개

Adobe Flash Player 화면에 텍스트를 표시하려면 TextField 클래스의 인스턴스를 사용합니다. TextField 클래스는 Adobe Flex 프레임워크 및 Flash 제작 환경에서 제공되는 기타 텍스트 기반 구성 요소(예: TextArea 구성 요소 또는 TextInput 구성 요소)의 기초가 됩니다. Flash 제작 환경에서의 텍스트 구성 요소 사용에 대한 자세한 내용은 *Flash 사용 설명서*의 "텍스트 제어"를 참조하십시오.

텍스트 필드 내용은 텍스트 파일이나 데이터베이스 같은 외부 소스에서 로드한 SWF 파일에 미리 지정하거나, 응용 프로그램을 사용하는 사용자가 직접 입력할 수 있습니다. 텍스트 필드에서는 텍스트가 렌더링된 HTML 내용으로 표시되고 렌더링된 HTML에 이미지를 포함할 수 있습니다. 텍스트 필드의 인스턴스를 구축한 경우에는 TextFormat 클래스와 StyleSheet 클래스처럼 flash.text 패키지 클래스를 사용하여 텍스트의 모양을 제어할 수 있습니다. flash.text 패키지에는 ActionScript에서 텍스트를 만들고, 관리하고, 서식을 지정하는 작업과 관련된 거의 모든 클래스가 들어 있습니다.

TextFormat 객체를 사용하여 서식을 정의한 다음 이 객체를 텍스트 필드에 할당하여 텍스트 서식을 지정할 수 있습니다. 텍스트 필드에 HTML 텍스트가 있으면 StyleSheet 객체를 텍스트 필드에 적용하여 텍스트 필드 내용의 특정 부분에 스타일을 지정할 수 있습니다. TextFormat 객체나 StyleSheet 객체에는 색, 크기 및 두께 등 텍스트의 모양을 정의하는 속성이 있습니다. TextFormat 객체는 텍스트 필드 내의 모든 내용에 속성을 할당하거나 일정 범위의 텍스트에 할당합니다. 예를 들어, 같은 텍스트 필드 내에서 한 문장은 빨간색 굵은 텍스트로, 다음 문장은 파란색 기울임꼴 텍스트로 표시할 수 있습니다.

텍스트 서식에 대한 자세한 내용은 488페이지의 "텍스트 서식 할당"을 참조하십시오.

텍스트 필드의 HTML 텍스트에 대한 자세한 내용은 482페이지의 "HTML 텍스트 표시"를 참조하십시오.

스타일 시트에 대한 자세한 내용은 489페이지의 "CSS(Cascading Style Sheet) 적용"을 참조하십시오.

flash.text 패키지의 클래스뿐 아니라 flash.events.TextEvent 클래스를 사용해서도 텍스트와 관련된 사용자 액션에 응답할 수 있습니다.

텍스트와 관련된 일반적인 작업

이 장에서는 다음과 같은 일반적인 텍스트 관련 작업이 설명됩니다.

- 텍스트 필드 내용 수정
- 텍스트 필드에 HTML 사용
- 텍스트 필드에 이미지 사용
- 텍스트 선택 및 사용자가 선택한 텍스트를 사용한 작업
- 텍스트 입력 캡처
- 텍스트 입력 제한
- 텍스트에 서식 및 CSS 스타일 적용
- 선명도, 두께 및 앤티앨리어싱으로 텍스트 표시 미세 조정
- ActionScript에서 정적 텍스트 필드 액세스 및 작업

중요한 개념 및 용어

다음 참조 목록에는 이 장에 나오는 중요한 용어에 대한 설명이 나와 있습니다.

- CSS(Cascading Style Sheet): XML 또는 HTML 형식으로 구성된 내용의 스타일과 서식 설정을 위한 표준 구문입니다.
- 장치 글꼴: 사용자 시스템에 설치되어 있는 글꼴입니다.
- 동적 텍스트 필드: 사용자 입력이 아닌 ActionScript에 의해 내용을 변경할 수 있는 텍스트 필드입니다.
- 포함 글꼴: 데이터가 포함된 글꼴로, 문자 외곽선 데이터는 응용 프로그램의 SWF 파일에 저장되어 있습니다.
- HTML 텍스트: ActionScript를 사용하여 텍스트 필드에 입력한 텍스트 내용으로, HTML 서식 태그와 실제 텍스트 내용이 들어 있는 텍스트입니다.
- 입력 텍스트 필드: 사용자 입력 또는 ActionScript에 의해 내용을 변경할 수 있는 텍스트 필드입니다.
- 정적 텍스트 필드: Flash 제작 도구에서 만들어졌으며 SWF 파일 실행 중에 내용을 변경할 수 없는 텍스트 필드입니다.
- 텍스트 행 메트릭: 텍스트 기준선, 문자의 높이, 디센더(일부 소문자에서 기준선 아래로 뻗은 부분) 크기 등, 텍스트 필드 내 텍스트 내용에 대한 다양한 부분의 크기 값입니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장은 ActionScript의 텍스트 필드 작업을 다루므로, 이 장에 나오는 모든 코드 샘플은 기본적으로 TextField 객체(Flash 제작 도구에서 생성하여 스테이지에 배치한 TextField 객체 또는 ActionScript를 사용하여 생성한 TextField 객체) 조작과 관련이 있습니다. 샘플을 테스트하려면 코드로 텍스트 필드를 조작한 결과를 Flash Player에서 확인해야 합니다.

이 장의 예제는 두 그룹으로 나누어 집니다. 첫 번째 예제 유형은 TextField 객체를 명시적으로 생성하지 않고 조작합니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [텍스트 도구]를 사용하여 스테이지에 동적 텍스트 필드를 작성합니다.
5. 텍스트 필드를 선택한 상태에서 속성 관리자에서 텍스트 필드에 인스턴스 이름을 지정합니다. 이 이름은 예제 코드 샘플에서 해당 텍스트 필드에 대해 사용한 이름과 일치해야 합니다. 예를 들어, 코드 샘플에서 `myTextField`라는 텍스트 필드를 조작하려는 경우 텍스트 필드 이름도 `myTextField`로 지정해야 합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   코드 샘플에 지정된 대로 텍스트 필드를 코드 조작한 결과가 화면에 표시됩니다.

이 장에 나오는 두 번째 유형의 예제 코드 샘플은 SWF에 대한 문서 클래스로 사용될 클래스 정의로 구성되어 있습니다. 이러한 샘플의 경우, 예제 코드에 의해 TextField 인스턴스가 만들어지므로 사용자가 인스턴스를 개별적으로 만들어야 합니다. 이러한 유형의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 새 ActionScript 파일을 만들어 위의 Flash 문서와 동일한 디렉토리에 저장합니다. 파일 이름은 코드 샘플에 있는 클래스의 이름과 일치해야 합니다. 예를 들어, 코드 샘플에 정의된 클래스 이름이 TextFieldTest인 경우 TextFieldTest.as라는 이름을 사용하여 ActionScript 파일을 저장합니다.
3. ActionScript 파일에 코드 샘플을 복사하고 파일을 저장합니다.
4. Flash 문서에서 스테이지 또는 작업 영역의 빈 부분을 클릭하여 문서 속성 관리자를 활성화합니다.
5. 텍스트에서 복사한 ActionScript 클래스의 이름을 속성 관리자의 [문서 클래스] 필드에 입력합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   화면에 표시되는 예제 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 “이 장에 제시된 예제 코드 샘플 테스트”에서 자세하게 설명합니다.

텍스트 표시

Adobe Flex Builder 및 Flash 제작 도구와 같은 제작 도구는 텍스트 관련 구성 요소나 텍스트 도구 등 텍스트를 표시할 수 있는 여러 옵션을 제공하지만 기본 방식은 텍스트 필드에서 프로그래밍 방식으로 텍스트를 표시하는 것입니다.

텍스트의 유형

텍스트 필드의 텍스트 유형은 해당 소스에 따라 다른 특징을 나타냅니다.

- 동적 텍스트

  동적 텍스트에는 텍스트 파일, XML 파일 또는 원격 웹 서비스와 같은 외부 소스에서 로드되는 내용이 포함됩니다. 자세한 내용은 481페이지의 “텍스트의 유형”을 참조하십시오.

- 입력 텍스트

  입력 텍스트는 사용자가 입력하는 텍스트 또는 사용자가 편집할 수 있는 동적 텍스트입니다. 입력 텍스트에 서식을 지정하려면 스타일 시트를 설정하거나 flash.text.TextFormat 클래스를 사용하여 입력 내용의 텍스트 필드에 속성을 할당할 수 있습니다. 자세한 내용은 486페이지의 “텍스트 입력 캡처”를 참조하십시오.

- 정적 텍스트

  정적 텍스트는 Flash 제작 도구로만 만들어집니다. ActionScript 3.0을 사용하는 경우 정적 텍스트 인스턴스를 만들 수 없습니다. 그러나 StaticText 및 TextSnapshot 같은 ActionScript 클래스를 사용하여 기존의 정적 텍스트 인스턴스를 조작할 수는 있습니다. 자세한 내용은 495페이지의 “정적 텍스트를 사용한 작업”을 참조하십시오.

텍스트 필드 내용 수정

`flash.text.TextField.text` 속성에 문자열을 할당하여 동적 텍스트를 정의할 수 있습니다. 다음과 같이 속성에 직접 문자열을 할당할 수 있습니다.

```
myTextField.text = “Hello World”;
```

다음 예제와 같이 `text` 속성에 스크립트에서 정의한 변수의 값을 할당할 수도 있습니다.

```
package
{
  import flash.display.Sprite;
  import flash.text.*;

  public class TextWithImage extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello World";

    public function TextWithImage()
```

```
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

그 외의 방법으로 원격 변수의 값을 `text` 속성에 할당할 수 있습니다. 원격 소스에서 텍스트 값을 로드하는 옵션은 세 가지입니다.

- flash.net.URLLoader 및 flash.net.URLRequest 클래스는 로컬 또는 원격 위치에서 텍스트의 변수를 로드합니다.
- `FlashVars` 속성은 SWF 파일을 호스팅하는 HTML 페이지에 포함되고, 텍스트 변수의 값을 포함할 수 있습니다.
- flash.net.SharedObject 클래스는 값의 지속적인 저장을 관리합니다. 자세한 내용은 625페이지의 "로컬 데이터 저장"을 참조하십시오.

HTML 텍스트 표시

flash.text.TextField 클래스에는 `htmlText` 속성이 있습니다. 이 속성을 사용하면 내용의 서식을 지정할 수 있도록 텍스트 문자열을 HTML 태그가 포함된 하나의 단위로 식별할 수 있습니다. 다음 예제와 같이 문자열 값을 Flash Player의 `htmlText` 속성(`text` 속성이 아님)에 할당하여 텍스트를 HTML로 렌더링해야 합니다.

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as
  <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player에서는 HTML 태그의 하위 집합과 `htmlText` 속성의 엔터티를 지원합니다. *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 `flash.text.TextField.htmlText` 속성 설명 부분에 지원되는 HTML 태그 및 엔터티가 자세히 설명되어 있습니다.

`htmlText` 속성을 사용하여 내용을 지정하면 스타일 시트나 `textformat` 태그를 사용하여 내용의 서식을 관리할 수 있습니다. 자세한 내용은 488페이지의 "텍스트 서식 지정"을 참조하십시오.

텍스트 필드에서 이미지 사용

내용을 HTML 텍스트로 표시할 때 얻을 수 있는 또 다른 장점은 텍스트 필드에 이미지를 포함할 수 있다는 것입니다. `img` 태그를 사용하여 로컬 또는 원격의 이미지를 참조하고, 연결된 텍스트 필드 내에 표시할 수 있습니다.

다음 예제에서는 `myTextBox`라는 텍스트 필드를 만들고 SWF 파일과 같은 디렉토리에 저장된 눈의 JPG 이미지를 표시된 텍스트 내에 포함합니다.

```
package
{
  import flash.display.Sprite;
  import flash.text.*;

  public class TextWithImage extends Sprite
  {
    private var myTextBox:TextField;
    private var myText:String = "<p>This is <b>some</b> content to
  <i>test</i> and <i>see</i></p><p><img src='eye.jpg' width='20'
  height='20'></p><p>what can be rendered.</p><p>You should see an eye
  image and some <u>HTML</u> text.</p>";

    public function TextWithImage()
    {
      myTextBox.width = 200;
      myTextBox.height = 200;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.border = true;

      addChild(myTextBox);
      myTextBox.htmlText = myText;
    }
  }
}
```

`img` 태그는 JPEG, GIF, PNG 및 SWF 파일을 지원합니다.

텍스트 필드에서 텍스트 스크롤

많은 경우에 텍스트가 텍스트를 표시하는 텍스트 필드보다 깁니다. 또는 사용자가 한 번에 표시할 수 있는 것보다 많은 텍스트를 입력할 수 있는 입력 필드가 있을 수 있습니다. flash.text.TextField 클래스의 스크롤 관련 속성을 사용하여 긴 내용을 가로 또는 세로로 관리할 수 있습니다.

스크롤 관련 속성에는 TextField.scrollV와 TextField.scrollH 그리고 maxScrollV와 maxScrollH가 있습니다. 이러한 속성을 사용하여 마우스 클릭이나 키 누르기 등의 이벤트에 응답할 수 있습니다.

다음 예제에서는 크기가 설정되어 있고 필드에서 한 번에 표시할 수 있는 것보다 많은 텍스트가 포함된 텍스트 필드를 만듭니다. 사용자가 텍스트 필드를 클릭하면 텍스트가 세로로 스크롤됩니다.

```
package
{
  import flash.display.Sprite;
  import flash.text.*;
  import flash.events.MouseEvent;

  public class TextScrollExample extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello world and welcome to the show. It's
  really nice to meet you. Take your coat off and stay a while. OK, show is
  over. Hope you had fun. You can go home now. Don't forget to tip your
  waiter. There are mints in the bowl by the door. Thank you. Please come
  again.";

    public function TextScrollExample()
    {
      myTextBox.text = myText;
      myTextBox.width = 200;
      myTextBox.height = 50;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.background = true;
      myTextBox.border = true;

      var format:TextFormat = new TextFormat();
      format.font = "Verdana";
      format.color = 0xFF0000;
      format.size = 10;

      myTextBox.defaultTextFormat = format;
      addChild(myTextBox);
      myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
    }
```

```
    public function mouseDownScroll(event:MouseEvent):void
    {
      myTextBox.scrollV++;
    }
  }
}
```

텍스트 선택 및 조작

동적 텍스트나 입력 텍스트를 선택할 수 있습니다. TextField 클래스의 텍스트 선택 속성과 메서드는 인덱스 위치를 사용하여 조작할 텍스트의 범위를 설정하므로 내용을 모르는 경우에도 프로그래밍 방식으로 동적 텍스트나 입력 텍스트를 선택할 수 있습니다.

예외 **Flash 제작 도구의 정적 텍스트 필드에서 선택 가능한 옵션을 선택하는 경우 표시 목록에 내보내 배치되는 텍스트 필드는 일반적인 정규 동적 텍스트 필드입니다.**

텍스트 선택

`flash.text.TextField.selectable` 속성은 기본적으로 `true`이며 `setSelection()` 메서드를 사용하여 프로그래밍 방식으로 텍스트를 선택할 수 있습니다.

예를 들어, 사용자가 텍스트 필드를 클릭할 때 텍스트 필드 내에 특정 텍스트가 선택되도록 설정할 수 있습니다.

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN
  ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
  myTextField.setSelection(49, 65);
}
```

이와 유사하게 텍스트 필드의 텍스트가 최초에 표시된 대로 선택되도록 하려면, 해당 텍스트 필드가 표시 목록에 추가될 때 호출되는 이벤트 핸들러 함수를 작성합니다.

사용자가 선택한 텍스트 캡처

TextField 클래스의 selectionBeginIndex 속성 및 selectionEndIndex 속성은 "읽기 전용"이므로 프로그래밍 방식으로 텍스트를 선택하도록 설정할 수 없으며, 현재 사용자가 선택한 항목을 캡처하는 데 사용할 수 있습니다. 또한 입력 텍스트 필드는 caretIndex 속성을 사용할 수 있습니다.

예를 들어, 다음 코드는 사용자가 선택한 텍스트의 인덱스 값을 추적합니다.

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index
  values for the first and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
  trace("First letter index position: " + myTextField.selectionBeginIndex);
  trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

TextFormat 객체 속성 컬렉션을 선택 항목에 적용하여 텍스트의 모양을 변경할 수 있습니다. 선택한 텍스트에 TextFormat 속성의 컬렉션을 적용하는 데 대한 자세한 내용은 492페이지의 "텍스트 필드의 텍스트 범위 서식 지정"을 참조하십시오.

텍스트 입력 캡처

기본적으로 텍스트 필드의 type 속성은 dynamic으로 설정되어 있습니다. TextFieldType 클래스를 사용하여 type 속성을 input으로 설정하면 사용자 입력을 수집하고 응용 프로그램의 다른 부분에 사용할 값을 저장할 수 있습니다. 입력 텍스트 필드는 사용자가 프로그램의 다른 위치에 사용할 텍스트 값을 정의할 수 있는 양식 및 응용 프로그램에 유용합니다.

예를 들어 다음 코드는 myTextBox라는 입력 텍스트 필드를 만듭니다. 사용자가 필드에 텍스트를 입력하면 textInput 이벤트가 트리거됩니다. textInputCapture라는 이벤트 핸들러는 입력한 텍스트 문자열을 캡처하고 이 문자열에 변수를 할당합니다. Flash Player에서는 myOutputBox라는 다른 텍스트 필드에 새 텍스트를 표시합니다.

```
package
{
  import flash.display.Sprite;
  import flash.display.Stage;
  import flash.text.*;
  import flash.events.*;

  public class CaptureUserInput extends Sprite
  {
```

```
    private var myTextBox:TextField = new TextField();
    private var myOutputBox:TextField = new TextField();
    private var myText:String = "Type your text here.";

    public function CaptureUserInput()
    {
      captureText();
    }

    public function captureText():void
    {
      myTextBox.type = TextFieldType.INPUT;
      myTextBox.background = true;
      addChild(myTextBox);
      myTextBox.text = myText;
      myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
    }

    public function textInputCapture(event:TextEvent):void
    {
      var str:String = myTextBox.text;
      createOutputBox(str);
    }

    public function createOutputBox(str:String):void
    {
      myOutputBox.background = true;
      myOutputBox.x = 200;
      addChild(myOutputBox);
      myOutputBox.text = str;
    }

  }
}
```

텍스트 입력 제한

입력 텍스트 필드는 응용 프로그램의 양식이나 대화 상자에 사용되는 경우가 많으므로 암호와 같이 사용자가 텍스트 필드에 입력할 수 있는 문자 유형을 제한하거나 텍스트를 숨길 수 있습니다. flash.text.TextField 클래스에는 사용자 입력을 제어하도록 설정할 수 있는 `displayAsPassword` 속성 및 `restrict` 속성이 있습니다.

`displayAsPassword` 속성은 사용자가 텍스트를 입력할 때 입력한 내용을 연속되는 별표로 표시하여 숨깁니다. `displayAsPassword`를 `true`로 설정하면 [잘라내기] 및 [복사] 명령과 해당 키보드 단축키가 작동하지 않습니다. 다음 예제와 같이 `displayAsPassword` 속성을 배경 및 색 등 다른 속성과 동일한 방식으로 할당합니다.

```
myTextBox.type = TextFieldType.INPUT;
```

```
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

restrict 속성의 경우 사용자가 입력 텍스트 필드에 입력할 수 있는 문자를 지정해야 하므로 좀 더 복잡합니다. 특정 글자, 숫자 또는 글자, 숫자, 문자의 범위를 허용할 수 있습니다. 다음 코드를 사용하면 사용자가 텍스트 필드에 대문자(숫자나 특수 문자 제외)만 입력할 수 있습니다.

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0에서는 범위를 정의할 하이픈 및 제외된 문자를 정의하는 캐럿을 사용합니다. 입력 텍스트 필드에 제한할 내용을 정의하는 데 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 flash.text.TextField.restrict 속성 항목을 참조하십시오.

텍스트 서식 지정

프로그래밍 방식으로 텍스트 서식을 지정하는 데에는 여러 가지 옵션이 있습니다. TextField 인스턴스에서 TextFIeld.thickness, TextField.textColor 및 TextField.textHeight 속성 등의 속성을 직접 설정할 수 있습니다. 또는 htmlText 속성을 사용하여 텍스트 필드의 내용을 지정하고 b, i 및 u 등 지원되는 HTML 태그를 사용할 수 있습니다. 그러나 TextFormat 객체를 일반 텍스트가 포함된 텍스트 필드에 적용하거나 StyleSheet 객체를 htmlText 속성이 포함된 텍스트 필드에 적용할 수도 있습니다. TextFormat 및 StyleSheet 객체를 사용하면 응용 프로그램 전체의 텍스트 모양을 가장 잘 제어할 수 있으며 일관된 모양이 됩니다. TextFormat 또는 StyleSheet 객체를 정의하고 해당 객체를 응용 프로그램의 여러 또는 모든 텍스트 필드에 적용할 수 있습니다.

텍스트 서식 할당

TextFormat 클래스를 사용하여 다양한 텍스트 표시 속성을 설정하고 이러한 속성을 TextField 객체의 전체 내용이나 일정 범위의 텍스트에 적용할 수 있습니다.

다음 예제에서는 TextFormat 객체를 전체 TextField 객체에 적용하고, 또 하나의 TextFormat 객체를 TextField 객체 내의 텍스트 범위에 적용합니다.

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
```

```
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

TextField.setTextFormat() 메서드는 텍스트 필드에 이미 표시되어 있는 텍스트에만 영향을 미칩니다. TextField의 내용이 변경되면 사용자의 응용 프로그램에서 TextField.setTextFormat() 메서드를 다시 호출하여 서식을 다시 적용해야 할 수 있습니다. TextField 객체의 defaultTextFormat 속성을 설정하여 사용자가 입력한 텍스트에 대해 이 서식이 사용되도록 지정할 수도 있습니다.

CSS(Cascading Style Sheet) 적용

텍스트 필드에는 일반 텍스트나 HTML 형식의 텍스트가 포함될 수 있습니다. 일반 텍스트는 인스턴스의 text 속성에 저장되고 HTML 텍스트는 htmlText 속성에 저장됩니다.

CSS 스타일 선언으로 텍스트 스타일을 정의하여 여러 가지 텍스트 필드에 적용할 수 있습니다. CSS 스타일 선언은 응용 프로그램 코드에 작성하거나 런타임 시 외부 CSS 파일에서 로드해 올 수 있습니다.

flash.text.StyleSheet 클래스는 CSS 스타일을 처리합니다. StyleSheet 클래스는 제한된 CSS 속성 집합을 인식합니다. StyleSheet 클래스가 지원하는 스타일 속성에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*의 flash.textStylesheet 항목을 참조하십시오.

다음 예제와 같이 코드에서 CSS 스타일을 만든 다음 StyleSheet 객체를 사용하여 HTML 텍스트에 이러한 스타일을 적용할 수 있습니다.

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

StyleSheet 객체 생성 후, 예제 코드는 스타일 선언 속성 집합을 포함하는 간단한 객체를 만듭니다. 그런 다음 StyleSheet.setStyle() 메서드를 호출하며 이 메서드는 ".darkred"라는 스타일 시트에 새 스타일을 추가합니다. 다음으로, StyleSheet 객체를 TextField 객체의 styleSheet 속성에 할당하여 스타일 시트 서식을 적용합니다.

CSS 스타일을 적용하려면 htmlText 속성이 설정되기 전에 스타일 시트가 TextField 객체에 적용되어야 합니다.

스타일 시트가 있는 텍스트 필드는 편집할 수 없도록 만들어져 있습니다. 입력 텍스트 필드가 있고 이 필드에 스타일 시트를 지정한 경우 텍스트 필드에 스타일 시트의 속성이 표시되기는 하지만 사용자가 이 텍스트 필드에 새 텍스트를 입력할 수는 없습니다. 또한 스타일 시트가 지정된 텍스트 필드에서는 다음과 같은 ActionScript API를 사용할 수 없습니다.

- `TextField.replaceText()` 메서드
- `TextField.replaceSelectedText()` 메서드
- `TextField.defaultTextFormat` 속성
- `TextField.setTextFormat()` 메서드

텍스트 필드에 스타일 시트를 지정했지만 나중에 `TextField.styleSheet` 속성을 `null`로 설정한 경우 `TextField.text` 및 `TextField.htmlText` 속성이 해당 내용에 태그와 속성을 추가하여 이전에 지정된 스타일 시트의 서식을 통합합니다. 원래의 `htmlText` 속성을 유지하려면 이 속성을 변수에 저장한 후에 스타일 시트를 `null`로 설정하십시오.

외부 CSS 파일 로드

CSS로 서식을 지정하는 방법은 런타임에 외부 파일에서 CSS 정보를 로드할 수 있는 경우 더욱 효과적입니다. CSS 데이터가 응용 프로그램 외부에 있으면 ActionScript 3.0 소스 코드를 변경하지 않고도 응용 프로그램에서 텍스트의 시각적 스타일을 변경할 수 있습니다. 응용 프로그램을 배포한 후에는 응용 프로그램의 SWF 파일을 다시 배포하지 않고도 외부 CSS 파일을 변경하여 응용 프로그램의 모양을 변경할 수 있습니다.

`StyleSheet.parseCSS()` 메서드는 CSS 데이터를 포함한 문자열을 StyleSheet 객체에서의 스타일 선언으로 변환합니다. 다음 예제는 외부 CSS 파일을 읽고 해당 스타일 선언을 TextField 객체에 적용하는 방법입니다.

우선 로드할 CSS 파일의 내용은 다음과 같습니다. 이 파일의 이름은 example.css로 지정됩니다.

```
p {
  font-family: Times New Roman, Times, _serif;
  font-size: 14;
}

h1 {
  font-family: Arial, Helvetica, _sans;
  font-size: 20;
  font-weight: bold;
}

.bluetext {
  color: #0000CC;
}
```

다음으로, example.css 파일을 로드하고 해당 스타일을 TextField 내용에 적용하는 클래스 ActionScript 코드가 있습니다.

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.net.URLLoader;
  import flash.net.URLRequest;
  import flash.text.StyleSheet;
  import flash.text.TextField;
  import flash.text.TextFieldAutoSize;

  public class CSSFormattingExample extends Sprite
  {
    var loader:URLLoader;
    var field:TextField;
    var exampleText:String = "<h1>This is a headline</h1>" +
      "<p>This is a line of text. <span class='bluetext'>" +
      "This line of text is colored blue.</span></p>";

    public function CSSFormattingExample():void
    {
      field = new TextField();
      field.width = 300;
      field.autoSize = TextFieldAutoSize.LEFT;
      field.wordWrap = true;
      addChild(field);

      var req:URLRequest = new URLRequest("example.css");

      loader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
      loader.load(req);
    }

    public function onCSSFileLoaded(event:Event):void
    {
      var sheet:StyleSheet = new StyleSheet();
      sheet.parseCSS(loader.data);
      field.styleSheet = sheet;
      field.htmlText = exampleText;
    }
  }
}
```

CSS 데이터가 로드되면 `onCSSFileLoaded()` 메서드는 `StyleSheet.parseCSS()` 메서드를 실행하고 호출하여 스타일 선언을 StyleSheet 객체에 전송합니다.

텍스트 필드의 텍스트 범위 서식 지정

flash.text.TextField 클래스의 메서드 중 특히 유용한 것은 `setTextFormat()` 메서드입니다. `setTextFormat()`을 사용하면 특정 속성을 텍스트 필드의 일부 내용에 할당하여 사용자 입력에 응답할 수 있습니다. 예를 들어, 특정 항목이 필수 입력 항목이라는 점을 사용자에게 알리는 양식 또는 사용자가 텍스트 일부를 선택했을 때 텍스트 필드 내에서 텍스트 단락의 특정 하위 섹션 강조를 변경하는 양식 등을 만들 수 있습니다.

다음 예제는 사용자가 텍스트 필드를 클릭하면 특정 문자 범위에 대해 `TextField.setTextFormat()`을 사용하여 `myTextField` 내용 일부의 모양을 변경합니다.

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN
  ALL CAPS changes format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
  myTextField.setTextFormat(myformat, 49, 65);
}
```

고급 텍스트 렌더링

ActionScript 3.0은 flash.text 패키지에 다양한 클래스를 제공하여 포함된 글꼴, 앤티앨리어싱, 알파 채널 컨트롤 및 기타 특정 설정 등 표시되는 텍스트의 속성을 제어합니다. *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에는 CSMSettings, Font, TextRenderer 클래스 등의 클래스와 그 속성이 자세히 설명되어 있습니다.

포함 글꼴 사용

응용 프로그램에서 TextField에 특정 글꼴을 지정하면 Flash Player에서는 이름이 같은 장치 글꼴(사용자의 컴퓨터에 있는 글꼴)을 검색합니다. 사용자의 시스템에 글꼴이 없거나 해당 이름과 조금 다른 글꼴이 있는 경우 텍스트가 원하는 모양과 아주 다르게 표시될 수 있습니다.

원하는 글꼴이 정확히 표시되게 하기 위해 응용 프로그램의 SWF 파일에 해당 글꼴을 포함시킬 수 있습니다. 포함 글꼴을 사용하면 다음과 같은 여러 가지 장점이 있습니다.

- 포함 글꼴 문자는 앤티앨리어싱을 통해 가장자리가 더 부드럽게 표시되므로 특히 텍스트 크기가 클 때 효과적입니다.

- 포함 글꼴을 사용하는 텍스트만 회전이 가능합니다.
- 포함 글꼴 텍스트는 투명 또는 반투명으로 설정할 수 있습니다.
- `kerning` CSS 스타일을 포함된 글꼴과 함께 사용할 수 있습니다.

포함 글꼴을 사용하는 경우에는 응용 프로그램의 파일 크기나 다운로드 크기가 증가한다는 것이 가장 큰 단점입니다.

사운드 파일을 응용 프로그램의 SWF 파일에 포함시키는 정확한 방법은 개발 환경에 따라 다릅니다.

글꼴을 포함한 후에는 TextField에서 정확한 포함 글꼴을 사용할 수 있습니다.

- TextField의 `embedFonts` 속성을 `true`로 설정합니다.
- TextFormat 객체를 만들어 그 `fontFamily` 속성을 포함된 글꼴 이름으로 설정합니다. 그런 다음, TextFormat 객체를 TextField에 적용합니다. 포함된 글꼴을 지정할 때 `fontFamily` 속성 이름이 하나여야만 하며, 여러 글꼴 이름을 쉼표로 구분한 목록을 사용할 수 없습니다.
- TextFields 또는 구성 요소에 글꼴을 설정하는 데 CSS 스타일을 사용하려는 경우 `font-family` CSS 속성을 포함된 글꼴 이름으로 설정합니다. 포함된 글꼴을 지정하려는 경우 `font-family` 속성 이름은 하나여야만 하며 이름 목록을 포함할 수 없습니다.

Flash에 글꼴 포함

Flash 제작 도구를 사용하면 트루타입 글꼴 및 Type 1 Postscript 글꼴을 포함하여, 시스템에 설치되어 있는 거의 모든 글꼴을 포함시킬 수 있습니다.

글꼴을 Flash 응용 프로그램에 포함할 수 있는 방법은 다음을 비롯해 여러 가지가 있습니다.

- 스테이지에서 TextField의 글꼴 및 스타일 속성을 선택하고 [글꼴 포함] 체크 상자를 클릭
- 글꼴 심볼 만들기 및 참조
- 포함된 글꼴 심볼이 포함된 런타임 공유 라이브러리를 만들어 사용

Flash 응용 프로그램에 글꼴을 포함하는 방법에 대한 자세한 내용은 *Flash 사용 설명서*의 "동적 또는 입력 텍스트 필드의 포함 글꼴"을 참조하십시오.

선명도, 두께 및 앤티앨리어싱 제어

기본적으로 Flash Player에서는 텍스트 크기를 변경하거나, 색을 변경하거나 텍스트를 다양한 배경에 표시할 때 선명도, 두께 및 앤티앨리어싱 등 텍스트 표시 컨트롤 설정을 결정합니다. 매우 작거나 매우 큰 텍스트가 있는 경우 또는 텍스트가 여러 가지 고유한 배경 위에 있는 경우 등 일부 경우에는 이러한 설정을 직접 제어할 수 있습니다. CSMSettings 클래스처럼 `flash.text.TextRenderer` 클래스 및 이와 연결된 클래스를 사용하여 Flash Player 설정을 덮어쓸 수 있습니다. 이러한 클래스를 사용하면 포함된 텍스트의 렌더링 품질을 정확히 제어할 수 있습니다. 포함된 글꼴에 대한 자세한 내용은 492페이지의 "포함 글꼴 사용"을 참조하십시오.

예외

선명도, 두께 또는 `gridFitType` 속성을 설정하거나 `TextRenderer.setAdvancedAntiAliasingTable()` 메서드를 사용하려면 `flash.text.TextField.antiAliasType` 속성에 `AntiAliasType.ADVANCED` 값(기본값)이 있어야 합니다.

다음 예제는 `myFont`라는 포함된 글꼴을 사용하여, 사용자 정의 CSM(continuous stroke modulation) 속성 및 서식을 표시된 텍스트에 적용합니다. 사용자가 표시된 텍스트를 클릭하면 사용자 정의 설정이 적용됩니다.

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
  var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
  var myAliasTable:Array = new Array(myAntiAliasSettings);
  TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
  TextColorType.DARK_COLOR, myAliasTable);
}
```

정적 텍스트를 사용한 작업

정적 텍스트는 Flash 제작 도구에서만 만들어집니다. ActionScript를 사용하면 정적 텍스트를 프로그래밍 방식으로 인스턴스화할 수 없습니다. 텍스트가 매우 짧고 동적 텍스트처럼 변경할 수 있도록 만들어지지 않은 경우에는 정적 텍스트가 유용합니다. 정적 텍스트를 Flash 제작 도구의 스테이지에 그려진 원이나 사각형 같은 그래픽 요소의 일종으로 생각하십시오. 정적 텍스트는 동적 텍스트보다 훨씬 제한되지만 ActionScript 3.0에서는 flash.text.StaticText 클래스를 사용하여 정적 텍스트의 속성 값을 읽을 수 있는 기능을 지원합니다.
flash.text.TextSnapshot 클래스를 사용하는 경우에도 정적 텍스트에서 값을 읽을 수 있습니다.

StaticText 클래스로 정적 텍스트 필드에 액세스

일반적으로 스테이지에 배치된 정적 텍스트 인스턴스와 상호 작용하려면 Flash 제작 도구의 [액션] 패널에서 flash.text.StaticText 클래스를 사용합니다. 정적 텍스트가 포함된 SWF 파일과 상호 작용하는 ActionScript 파일에서 작업할 수도 있습니다. 두 경우 모두 정적 텍스트 인스턴스를 프로그래밍 방식으로 인스턴스화할 수는 없습니다. 정적 텍스트는 Flash CS3 제작 도구에서 만들어집니다.

ActionScript 3.0에서 기존 정적 텍스트 필드에 대한 참조를 만들려면 표시 목록에 있는 항목을 반복하고 변수를 할당합니다. 예를 들면 다음과 같습니다.

```
for (var i = 0; i < this.numChildren; i++) {
    var displayitem:DisplayObject = this.getChildAt(i);
    if (displayitem instanceof StaticText) {
        trace("a static text field is item " + i + " on the display list");
     var myFieldLabel:StaticText = StaticText(displayitem);
     trace("and contains the text: " + myFieldLabel.text);
    }
}
```

기존 정적 텍스트 필드에 대한 참조를 만들고 나면 이 필드의 속성을 ActionScript 3.0에서 사용할 수 있습니다. 타임라인의 프레임에 다음 코드를 추가하고, `myFieldLabel`이라는 변수가 정적 텍스트 참조에 할당되어 있다고 가정해 봅니다. 예제에서 `myField`라는 동적 텍스트 필드는 `myFieldLabel`의 `x` 및 `y` 값과 관련된 위치에 지정되고 `myFieldLabel`의 값을 다시 표시합니다.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

TextSnapshot 클래스 사용

기존의 정적 텍스트 인스턴스를 프로그래밍 방식으로 사용하려는 경우 flash.text.TextSnapshot 클래스를 사용하여 flash.display.DisplayObjectContainer의 `textSnapshot` 속성을 사용할 수 있습니다. 즉, `DisplayObjectContainer.textSnapshot` 속성에서 TextSnapshot 인스턴스를 만듭니다. 그러면 해당 인스턴스에 메서드를 적용하여 값을 가져오거나 정적 텍스트의 일부를 선택할 수 있습니다.

예를 들어, Stage에 "TextSnapshot Example" 텍스트가 포함된 정적 텍스트 필드를 배치합니다. 타임라인의 프레임 1에 다음 ActionScript를 추가합니다.

```
var mySnap:TextSnapshot = this.getTextSnapshot();
var count:Number = mySnap.getCount();
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

TextSnapshot 클래스는 텍스트를 응용 프로그램의 다른 부분에 있는 값으로 사용하려는 경우 로드된 SWF의 정적 텍스트 필드에서 텍스트를 가져올 때 유용합니다.

예제: 신문 스타일 텍스트 서식

신문 레이아웃 예제는 신문 기사 스타일로 텍스트 서식을 지정합니다. 기사의 제목, 부제, 본문 등이 입력 텍스트가 될 수 있습니다. 표시 너비와 높이가 지정되어 있다면 이 신문 레이아웃 예제에서는 제목과 부제의 서식은 표시 영역의 너비에 가득 차도록 설정될 것입니다. 기사 텍스트는 둘 이상의 단으로 표시됩니다.

이 예제에서는 다음과 같은 ActionScript 프로그래밍 기술을 보여 줍니다.

- TextField 클래스 확장
- 외부 CSS 파일 로드 및 적용
- CSS 스타일을 TextFormat 객체로 변환
- TextLineMetrics 클래스를 사용하여 텍스트 표시 크기 정보 가져오기

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. News Layout 응용 프로그램 파일은 Samples/NewsLayout 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
NewsLayout.mxml 또는 NewsLayout.fla	Flex(MXML) 또는 Flash(FLA)용 응용 프로그램의 사용자 인터페이스입니다.
StoryLayout.as	표시할 뉴스 기사의 모든 구성 요소를 배열하는 기본 ActionScript 클래스입니다.
FormattedTextField.as	자체 TextFormat 객체를 관리하는 TextField 클래스의 하위 클래스입니다.
HeadlineTextField.as	글꼴 크기를 원하는 너비에 맞게 조정하는 FormattedTextField 클래스의 하위 클래스입니다.
MultiColumnTextField.as	텍스트를 두 개 이상의 단으로 나누는 ActionScript 클래스입니다.
story.css	레이아웃의 텍스트 스타일을 정의하는 CSS 파일입니다.
newsconfig.xml	기사 내용이 포함된 XML 파일입니다.

외부 CSS 파일 읽기

신문 레이아웃 응용 프로그램은 로컬 XML 파일에서 기사 텍스트를 읽는 것으로 시작합니다. 그런 다음 제목, 부제 및 본문 텍스트의 서식 정보를 제공하는 외부 CSS 파일을 읽습니다.

CSS 파일은 기사의 표준 단락 스타일, 제목의 h1 및 부제의 h2 스타일 등 세 가지 스타일을 정의합니다.

```
p {
  font-family: Georgia, Times New Roman, Times, _serif;
  font-size: 12;
  leading: 2;
  text-align: justify;
}

h1 {
  font-family: Verdana, Arial, Helvetica, _sans;
  font-size: 20;
  font-weight: bold;
  color: #000099;
  text-align: left;
}
```

```
h2 {
  font-family: Verdana, Arial, Helvetica, _sans;
  font-size: 16;
  font-weight: normal;
  text-align: left;
}
```

외부 CSS 파일을 읽는 방법은 490페이지의 "외부 CSS 파일 로드"에 설명된 방법과 동일합니다. CSS 파일이 로드되면 응용 프로그램은 다음과 같이 `onCSSFileLoaded()` 메서드를 실행합니다.

```
public function onCSSFileLoaded(event:Event):void
{
  this.sheet = new StyleSheet();
  this.sheet.parseCSS(loader.data);

  h1Format = getTextStyle("h1", this.sheet);
  if (h1Format == null)
  {
    h1Format = getDefaultHeadFormat();
  }
  h2Format = getTextStyle("h2", this.sheet);
  if (h2Format == null)
  {
    h2Format = getDefaultHeadFormat();
    h2Format.size = 16;
  }
  displayStory();
}
```

`onCSSFileLoaded()` 메서드는 새 StyleSheet 객체를 만들고 이 객체가 입력 CSS 데이터를 파싱하도록 합니다. 기사의 본문 텍스트가 StyleSheet 객체를 직접 사용할 수 있는 MultiColumnTextField 객체에 표시됩니다. 헤드라인 필드는 서식 지정에 TextFormat 객체를 사용하는 HeadlineTextField 클래스를 사용합니다.

`onCSSFileLoaded()` 메서드는 두 HeadlineTextField 객체가 각각 `getTextStyle()` 메서드를 두 번 호출하여 CSS 스타일 선언을 TextFormat 객체로 변환합니다. `getTextStyle()` 메서드는 아래와 같습니다.

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
  var format:TextFormat = null;

  var style:Object = ss.getStyle(styleName);
  if (style != null)
  {
    var colorStr:String = style.color;
    if (colorStr != null && colorStr.indexOf("#") == 0)
    {
```

```
      style.color = colorStr.substr(1);
    }
    format = new TextFormat(style.fontFamily,
                style.fontSize,
                style.color,
                (style.fontWeight == "bold"),
                (style.fontStyle == "italic"),
                (style.textDecoration == "underline"),
                style.url,
                style.target,
                style.textAlign,
                style.marginLeft,
                style.marginRight,
                style.textIndent,
                style.leading);

    if (style.hasOwnProperty("letterSpacing"))
    {
      format.letterSpacing = style.letterSpacing;
    }
  }
  return format;
}
```

속성 이름과 속성 값의 의미는 CSS 스타일 선언과 TextFormat 객체가 서로 다릅니다. `getTextStyle()` 메서드는 CSS 속성 값을 TextFormat 객체에서 예상하는 값으로 변환합니다.

기사 구성 요소를 페이지에 배치

StoryLayout 클래스는 제목, 부제 및 본문 텍스트 필드의 서식과 레이아웃을 신문 스타일로 배열합니다. `displayText()` 메서드는 여러 필드를 처음에 만들어 배치합니다.

```
public function displayText():void
{
  headlineTxt = new HeadlineTextField(h1Format);
  headlineTxt.wordWrap = true;
  this.addChild(headlineTxt);
  headlineTxt.width = 600;
  headlineTxt.height = 100;
  headlineTxt.fitText(this.headline, 1, true);

  subtitleTxt = new HeadlineTextField(h2Format);
  subtitleTxt.wordWrap = true;
  subtitleTxt.y = headlineTxt.y + headlineTxt.height;
  this.addChild(subtitleTxt);
  subtitleTxt.width = 600;
  subtitleTxt.height = 100;
  subtitleTxt.fitText(this.subtitle, 1, false);
```

```
    storyTxt = new MultiColumnTextField(2, 10, 600, 200);
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 4;
    this.addChild(storyTxt);
    storyTxt.styleSheet = this.sheet;
    storyTxt.htmlText = loremIpsum;
}
```

각 필드의 y 속성은 이전 필드의 y 속성에 이전 필드 높이를 더한 값으로 설정되어 필드가 이전 필드 아래에 배치됩니다. HeadlineTextField 객체와 MultiColumnTextField 객체는 내용에 맞게 높이를 변경할 수 있으므로 이러한 동적 배치 계산이 필요합니다.

필드 크기에 맞게 글꼴 크기 변경

너비(픽셀) 및 표시할 최대 행 수가 지정되면 HeadlineTextField에서 글꼴 크기를 변경하여 텍스트를 필드에 맞춥니다. 텍스트가 짧으면 글꼴 크기가 아주 커져서 타블로이드 스타일의 제목이 만들어집니다. 마찬가지로 텍스트가 길면 글꼴 크기는 더 작아집니다.

다음 HeadlineTextField.fitText() 메서드는 글꼴 크기를 조정합니다.

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean =
    false, targetWidth:Number = -1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar
    * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // 포인트 크기가 너무 작습니다.
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
```

```
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // 마지막 크기로 다시 설정합니다 .
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

HeadlineTextField.fitText() 메서드는 간단한 재귀적인 방법을 사용하여 글꼴 크기를 조절합니다. 메서드는 먼저, 텍스트에서 문자당 평균 픽셀 수를 추측하고 이 값을 기반으로 시작점 크기를 계산합니다. 그런 다음 텍스트 필드의 글꼴 크기를 변경하고, 텍스트 줄 바꿈으로 인해 최대 텍스트 행 수보다 많은 행이 만들어졌는지 확인합니다. 행이 너무 많은 경우, shrinkText() 메서드를 호출하여 글꼴 크기를 줄인 다음 다시 시도합니다. 행이 많지 않은 경우에는 growText() 메서드를 호출하여 글꼴 크기를 늘린 다음 다시 시도합니다. 글꼴 크기를 한 포인트 늘릴 때 행이 너무 많이 만들어질 경우 그 지점에서 프로세스가 중단됩니다.

여러 열에 텍스트 분할

MultiColumnTextField 클래스는 텍스트를 여러 TextField 객체로 분산하여 텍스트를 신문 단처럼 배열합니다.

MultiColumnTextField() 생성자는 다음 예처럼 열당 하나씩 TextField 객체 배열을 만듭니다.

```
for (var i:int = 0; i < cols; i++)
{
  var field:TextField = new TextField();
  field.autoSize = TextFieldAutoSize.NONE;
  field.wordWrap = true;
  field.styleSheet = this.styleSheet;

  this.fieldArray.push(field);
  this.addChild(field);
}
```

addChild() 메서드를 사용하여 각 TextField 객체가 배열 및 표시 목록에 추가됩니다.

StoryLayout 객체의 text 속성 또는 styleSheet 속성이 변할 때마다 layoutColumns() 메서드가 호출되어 텍스트를 다시 표시합니다. layoutColumns() 메서드는 다음과 같이 getOptimalHeight() 메서드를 호출하여 주어진 레이아웃 폭에 모든 텍스트를 맞추기 위해 필요한 정확한 높이(픽셀)를 계산합니다.

```
public function getOptimalHeight(str:String):int
{
  if (fieldArray.length == 0 || str == "" || str == null)
  {
    return this.preferredHeight;
  }
  else
  {
    var colWidth:int = Math.floor( (this.preferredWidth -
      ((this.numColumns - 1) * gutter)) / this.numColumns);

    var field:TextField = fieldArray[0] as TextField;
    field.width = colWidth;
    field.htmlText = str;
```

```
        var linesPerCol:int = Math.ceil(field.numLines / this.numColumns);
        var metrics:TextLineMetrics = field.getLineMetrics(0);
        var prefHeight:int = linesPerCol * metrics.height;
        return prefHeight + 4;
    }
}
```

getOptimalHeight() 메서드는 먼저 각 열의 폭을 계산한 다음, 배열 내의 첫 번째 TextField 객체의 폭 및 htmlText 속성을 설정합니다. getOptimalHeight() 메서드는 이 첫 번째 TextField 객체를 사용하여 텍스트 내의 총 줄 바꿈 행 수를 알아낸 후 이 값을 기반으로 각 열에 필요한 행 수를 파악합니다. 그런 다음, TextField.getLineMetrics() 메서드를 호출하여 첫 번째 행 텍스트의 크기에 대한 자세한 내용이 포함된 TextLineMetrics 객체를 검색합니다. TextLineMetrics.height 속성은 어센트, 디센트, 행간을 포함한 텍스트 행의 전체 길이를 픽셀 단위로 나타냅니다. MultiColumnTextField 객체에 대한 최적의 높이는 행 높이에 열당 행 개수를 곱한 다음 4를 더한(TextField 객체 위아래 테두리 각각에 2픽셀씩 해당) 값이 됩니다.

다음은 layoutColumns() 메서드의 전체 코드입니다.

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    if (this.fitToText)
    {
        this.preferredHeight = this.getOptimalHeight(this._text);
    }

    var colWidth:int = Math.floor( (this.preferredWidth -
        ((numColumns - 1) * gutter)) / numColumns);
    var field:TextField;
    var remainder:String = this._text;
    var fieldText:String = "";

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;
        field.width = colWidth;
        field.height = this.preferredHeight;

        field.x = i * (colWidth + gutter);
        field.y = 0;

        field.htmlText = "<p>" + remainder + "</p>";

        remainder = "";
```

```
      fieldText = "";

      var linesRemaining:int = field.numLines;
      var linesVisible:int = field.numLines - field.maxScrollV + 1;
      for (var j:int = 0; j < linesRemaining; j++)
      {
        if (j < linesVisible)
        {
          fieldText += field.getLineText(j);
        }
        else
        {
          remainder +=  field.getLineText(j);
        }
      }

      field.htmlText = "<p>" + fieldText + "</p>";
    }
}
```

`getOptimalHeight()` 메서드를 호출하여 `preferredHeight` 속성을 설정한 다음, `layoutColumns()` 메서드는 TextField 객체를 반복하여 각 높이를 `preferredHeight` 값으로 설정합니다. 그런 다음 `layoutColumns()` 메서드는 각 필드에서 스크롤이 발생하지 않도록, 그리고 각 연속 필드의 텍스트가 이전 필드 텍스트 종료 지점에서 시작되도록 적절한 개수의 텍스트 행을 각 필드에 배분합니다.

제 18 장

18 비트맵을 사용한 작업

ActionScript 3.0에서는 벡터 드로잉 기능 이외에도 비트맵 이미지 만들기 기능 또는 SWF로 로드되는 외부 비트맵 이미지의 픽셀 데이터 조작 기능을 제공합니다. 개별 픽셀 값에 액세스하고 변경할 수 있으므로 통해 필터와 유사한 고유 이미지 효과를 연출하고 내장 노이즈 함수를 사용하여 텍스처 및 불규칙한 노이즈를 만들 수 있습니다. 이 장에서는 이러한 모든 기술에 대해 설명합니다.

목차

비트맵 작업의 기초

비트맵 작업 소개

디지털 이미지를 사용하여 작업할 때 보통 두 가지의 기본 그래픽 유형인 비트맵 그래픽과 벡터 그래픽을 접하게 됩니다. 래스터 그래픽이라고도 하는 비트맵 그래픽은 사각형 격자 구조로 배열된 작은 정사각형(픽셀)으로 구성되며, 벡터 그래픽은 선, 곡선 및 다각형 등과 같이 수학적으로 생성된 기하학적 형태로 구성됩니다.

비트맵 이미지는 픽셀 단위로 측정된 폭 및 높이와 각 픽셀의 비트 수(각 픽셀에 포함시킬 수 있는 색상 수)로 정의됩니다. RGB 색상 모델을 사용하는 비트맵 이미지의 경우 픽셀은 빨강, 녹색, 파랑이라는 세 개의 바이트로 구성되며 각 바이트는 0~255 범위의 값을 가집니다. 픽셀 내에서 바이트가 결합되면 미술가가 혼합한 페인트 색과 유사한 새로운 색상이 만들어집니다. 예를 들어, 바이트 값이 각각 빨강-255, 녹색-02 및 파랑-0일 경우 강렬한 주황색의 픽셀이 만들어집니다.

비트맵 이미지의 품질은 이미지의 해상도와 해당 색상 심도 비트 값을 결합한 결과에 따라 결정됩니다. *해상도*는 이미지 내에 포함된 픽셀 수와 관련이 있습니다. 따라서 픽셀 수가 많을수록 해상도가 높아지며 이미지는 선명해집니다. *색상 심도*는 한 픽셀에 포함될 수 있는 정보의 양과 관련이 있습니다. 예를 들어, 픽셀당 색상 심도 값이 16비트인 이미지의 경우 색상 심도가 48비트인 이미지와 동일한 색상 수를 표현할 수 없습니다. 따라서 48비트 이미지의 음영은 16비트 이미지보다 훨씬 매끄럽게 표현됩니다.

비트맵 그래픽은 해상도의 영향을 받기 때문에 크기 조절이 원활하게 이루어지지 않습니다. 특히 비트맵 이미지의 크기를 확장하게 되면 이러한 단점이 두드러집니다. 일반적으로 비트맵의 크기를 확장하면 디테일과 품질이 떨어집니다.

비트맵 파일 포맷

비트맵 이미지는 일반적인 몇 가지 파일 포맷으로 그룹화됩니다. 이러한 형식은 각기 다른 압축 알고리즘 유형을 사용하여 파일 크기를 줄이며 이미지의 최종 목적에 적합한 방식으로 이미지 품질을 최적화합니다. Adobe Flash Player에서 지원하는 비트맵 이미지 포맷은 GIF, JPG 및 PNG입니다.

GIF

GIF(Graphics Interchange Format)는 256 색상(8비트 색상) 이미지를 전송하기 위한 수단으로 CompuServe에서 1987년에 처음 개발된 포맷으로, 작은 파일 크기를 제공하므로 웹 기반 이미지에 적합합니다. 그러나 색상 팔레트가 한정되어 있기 때문에 GIF 이미지는 일반적으로 높은 수준의 음영과 색상 그래디언트를 요구하는 사진에는 대체로 적합하지 않습니다. GIF 이미지는 색상을 투명하게 매핑하도록 하는 단일 비트 투명도를 허용합니다. 따라서 웹 페이지 배경색은 투명도가 매핑된 이미지를 통해 표시됩니다.

JPEG

JPEG(Joint Photographic Experts Group)에서 개발한 JPEG(또는 JPG) 이미지 포맷은 손실 압축 알고리즘을 사용하여 작은 파일 크기의 24비트 색상 심도를 구현합니다. 손실 압축이란 이미지를 저장할 때마다 이미지의 품질과 데이터는 손실되지만 파일 크기가 줄어드는 것을 의미합니다. JPEG 포맷은 무수한 색상을 표시할 수 있기 때문에 사진에 적합합니다. 아울러 이미지에 적용할 압축 수준을 제어할 수 있어 이미지 품질 및 파일 크기를 조작할 수 있습니다.

PNG

PNG(Portable Network Graphics) 포맷은 특허받은 GIF 파일 포맷에 대한 오픈 소스 대안으로 개발되었으며, PNG는 최대 64비트 색상 심도를 지원하여 천육백만 색상 표현을 가능하게 합니다. PNG는 앞서 설명한 두 포맷보다 상대적으로 최근에 개발된 포맷이기 때문에 일부 구 버전의 브라우저에서는 PNG 파일을 지원하지 않습니다. JPG와는 다르게 PNG는 손실 없는 압축 방법을 사용하므로 이미지를 저장하더라도 이미지 데이터가 손실되지 않습니다. PNG 파일은 또한 최대 256 레벨의 투명도를 가능하게 하는 알파 투명도를 지원합니다.

투명 비트맵 및 불투명 비트맵

GIF 또는 PNG 포맷을 사용하는 비트맵 이미지의 경우 각 픽셀에 바이트(알파 채널)를 추가할 수 있습니다. 이 추가 픽셀 바이트는 픽셀의 투명도 값을 나타냅니다.

GIF 이미지는 단일 비트 투명도를 지원하므로 256 색상 팔레트에서 단 한 개의 색상만 투명으로 지정할 수 있습니다. 반면 PNG 이미지의 경우 최대 256 레벨의 투명도를 표현할 수 있습니다. 이 기능은 이미지 또는 텍스트를 배경에 블렌드해야 할 때 특히 유용합니다.

ActionScript 3.0에서는 BitmapData 클래스 내에서 이 추가 투명도 픽셀 바이트를 복제합니다. `BitmapDataChannel.ALPHA` 상수는 PNG 투명도 모델과 유사하며, 최대 256 레벨의 투명도를 제공합니다.

비트맵과 관련된 일반적인 작업

다음은 ActionScript에서 비트맵 이미지를 사용하여 작업할 때 수행할 수 있는 몇 가지 작업입니다.

- 스크린에 비트맵 표시
- 픽셀 색상 값 검색 및 설정
- 비트맵 데이터 복사
  - 비트맵과 정확히 일치하는 복사본 만들기
  - 비트맵의 한 색상 채널에서 다른 비트맵의 한 색상 채널로 데이터 복사
  - 스크린 표시 객체의 스냅샷을 비트맵으로 복사
- 비트맵 이미지에서 노이즈 및 텍스처 만들기
- 비트맵 스크롤

중요한 개념 및 용어

다음 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 알파: 색상 또는 이미지의 투명도 수준(보다 정확히는 불투명도)입니다. 알파의 양은 흔히 *알파 채널* 값으로 설명됩니다.
- ARGB 색상: 각 픽셀의 색상이 빨강, 녹색 및 파랑 색상 값의 혼합인 색상 체계로 투명도가 알파 값으로 지정됩니다.
- 색상 채널: 색상은 일반적으로 몇 가지 기본 색상(컴퓨터 그래픽의 경우 대개 빨강, 녹색, 파랑)의 혼합으로 표현됩니다. 각각의 기본 색상은 색상 채널로 간주되며 각 색상 채널의 색상 양이 서로 혼합되어 최종 색상을 만듭니다.
- 색상 심도: *비트 심도*라고도 하며, 각 픽셀에 할당된 컴퓨터 메모리 크기를 나타냅니다. 색상 심도는 이미지에서 표현할 수 있는 색상 수를 결정합니다.
- 픽셀: 비트맵 이미지의 최소 정보 단위로, 본질적으로는 색상 도트를 나타냅니다.
- 해상도: 이미지의 픽셀 크기로, 이미지에 포함된 세부 요소의 세분화 수준을 결정합니다. 해상도는 보통 픽셀 수를 단위로 하는 폭 및 높이로 표시됩니다.
- RGB 색상: 각 픽셀의 색상이 빨강, 녹색 및 파랑 색상 값의 혼합으로 표현되는 색상 체계를 의미합니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드를 직접 테스트할 수 있습니다. 이 장에서는 시각적 내용의 작성 및 조작을 다루므로 코드 목록을 테스트하려면 해당 코드를 실행한 후 작성된 SWF 파일에서 결과를 확인해야 합니다.

이 장의 코드 예제를 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. [타임라인]에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드를 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   작성된 SWF 파일에서 코드의 결과를 확인합니다.

대부분의 예제 코드에는 비트맵 이미지를 작성하는 코드가 포함되어 있기 때문에 비트맵 내용을 제공하지 않고도 직접 코드를 테스트할 수 있습니다. 사용자가 직접 만든 이미지에 대해 예제 코드를 테스트하려는 경우에는, 해당 이미지를 Adobe Flash CS3 Professional로 가져오거나 외부 이미지를 테스트용 SWF 파일에 로드한 후 이미지의 비트맵 데이터와 예제 코드를 함께 사용할 수 있습니다. 외부 이미지 로드에 대한 자세한 내용은 394페이지의 "표시 내용을 동적으로 로드"를 참조하십시오.

Bitmap 클래스 및 BitmapData 클래스

비트맵 이미지를 사용한 작업에 사용되는 기본 ActionScript 3.0 클래스에는 비트맵 이미지를 스크린에 표시하는 데 사용되는 Bitmap 클래스와 비트맵의 원시 이미지 데이터에 액세스하여 조작하는 데 사용되는 BitmapData 클래스가 있습니다.

Bitmap 클래스의 이해

DisplayObject 클래스의 하위 클래스인 Bitmap 클래스는 비트맵 이미지를 표시하는 데 사용되는 ActionScript 3.0의 기본 클래스입니다. 이러한 이미지는 flash.display.Loader 클래스를 통해 Flash로 로드되었거나 `Bitmap()` 생성자를 사용하여 동적으로 생성되었을 수 있습니다. 외부 소스에서 이미지를 로드할 경우 Bitmap 객체는 GIF, JPEG 또는 PNG 포맷의 이미지만 사용할 수 있습니다. 인스턴스화된 Bitmap 인스턴스는 Stage로 렌더링해야 할 BitmapData 객체의 래퍼로 간주할 수 있습니다. Bitmap 인스턴스는 표시 객체이므로 표시 객체의 모든 특징 및 기능을 Bitmap 인스턴스 조작에도 사용할 수 있습니다. 표시 객체 작업에 대한 자세한 내용은 349페이지의 제12장, “디스플레이 프로그래밍”을 참조하십시오.

픽셀에 물리기 및 픽셀 다듬기

모든 표시 객체에 공통적으로 사용할 수 있는 기능 이외에도 Bitmap 클래스는 비트맵 이미지에만 사용할 수 있는 추가 기능을 제공합니다.

Bitmap 클래스의 `pixelSnapping` 속성은 Flash 제작 도구의 픽셀에 물리기 기능과 유사하며 Bitmap 객체를 가장 가까운 픽셀에 물릴지 여부를 결정합니다. 이 속성은 또한 PixelSnapping 클래스에 정의된 세 가지 상수(`ALWAYS`, `AUTO`, `NEVER`)를 사용합니다.

픽셀 물리기 기능을 적용하는 구문은 다음과 같습니다.

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

비트맵 이미지의 배율을 조절하면 이미지가 흐려지거나 왜곡되는 경우가 많습니다. 이러한 왜곡 현상을 줄이기 위해 BitmapData 클래스의 `smoothing` 속성이 사용됩니다. 이 부울 속성이 `true`로 설정된 경우, 이미지 배율을 조절할 때 이미지의 픽셀이 앤티앨리어싱 처리되어 다듬어집니다. 그러면 이미지의 모양이 더 뚜렷하고 자연스러워집니다.

BitmapData 클래스의 이해

flash.display 패키지에 포함된 BitmapData 클래스는 로드되었거나 동적으로 생성된 비트맵 이미지에 포함된 픽셀의 사진 같은 스냅샷에 비유할 수 있습니다. 이 스냅샷은 객체 내 픽셀 데이터 배열로 표현됩니다. BitmapData 클래스에는 픽셀 데이터를 만들고 조작하는 데 유용한 일련의 내장 메서드도 포함되어 있습니다.

BitmapData 객체를 인스턴스화하려면 다음 코드를 사용하십시오.

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number,
  transparent:Boolean, fillColor:uinit);
```

`width` 매개 변수 및 `height` 매개 변수는 비트맵 크기를 지정하며, 두 매개 변수의 최대값은 각각 2880픽셀입니다. `transparent` 매개 변수는 비트맵 데이터에 알파 채널이 포함되어 있는지(`true`) 또는 포함되어 있지 않은지(`false`) 여부를 지정합니다. `fillColor` 매개 변수는 32비트 색상 값으로 배경색은 물론 투명도 값(`true`로 설정된 경우)을 지정합니다. 다음 예제에서는 투명도 50%의 주황색 배경을 가진 BitmapData 객체를 만듭니다.

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

새로 만든 BitmapData 객체를 스크린에 렌더링하려면 해당 객체를 Bitmap 인스턴스에 할당하거나 그 안에 래핑하십시오. 이를 위해 BitmapData 객체를 Bitmap 객체 생성자의 매개 변수로 전달하거나 기존 Bitmap 인스턴스의 `bitmapData` 속성에 할당할 수 있습니다. 또한, Bitmap 인스턴스를 포함할 표시 객체 컨테이너의 `addChild()` 또는 `addChildAt()` 메서드를 호출하여 표시 목록에 Bitmap 인스턴스를 추가해야 합니다. 표시 목록을 사용한 작업에 대한 자세한 내용은 360페이지의 “표시 목록에 표시 객체 추가”를 참조하십시오.

다음 예제에서는 빨간색으로 칠한 BitmapData 객체를 만든 후 해당 객체를 Bitmap 인스턴스에 표시합니다.

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false,
  0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

픽셀 조작

BitmapData 클래스에는 픽셀 데이터 값을 조작할 수 있는 메서드 집합이 포함되어 있습니다.

개별 픽셀 조작

픽셀 수준에서 비트맵 이미지의 모양을 바꿀 때에는 먼저 조작할 영역 내에 포함된 픽셀의 색상 값을 확인해야 합니다. 이러한 픽셀 값은 `getPixel()` 메서드를 사용하여 확인할 수 있습니다.

getPixel() 메서드는 매개 변수로 전달된 x, y(픽셀) 좌표에서 RGB 값을 가져옵니다. 조작하려는 픽셀에 투명도(알파 채널) 정보가 포함된 경우에는 getPixel32() 메서드를 사용해야 합니다. 이 메서드도 RGB 값을 가져오지만 getPixel()과는 다릅니다. getPixel32()에는 선택된 픽셀의 알파 채널(투명도) 값을 나타내는 추가 데이터가 포함되어 있습니다.

또는 비트맵에 포함된 픽셀의 색상이나 투명도만 변경하려는 경우에는 setPixel() 또는 setPixel32() 메서드를 사용할 수 있습니다. 픽셀 색상을 설정하려면 x, y 좌표 및 색상 값을 이러한 메서드 중 하나에 전달하기만 하면 됩니다.

다음 예제에서는 setPixel() 메서드를 사용하여 녹색 BitmapData 배경에 십자가를 그립니다. 그런 다음 getPixel() 메서드를 사용하여 좌표 50, 50에 있는 픽셀의 색상 값을 가져오고 반환된 값을 추적합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
  var red:uint = 0xFF0000;
  myBitmapData.setPixel(50, i, red);
  myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

단일 픽셀이 아닌 픽셀 그룹의 값을 읽으려는 경우에는 getPixels() 메서드를 사용합니다. 이 메서드는 매개 변수로 전달된 픽셀 데이터의 사각형 영역에서 바이트 배열을 생성합니다. 각각의 바이트 배열 요소(즉, 픽셀 값)는 부호 없는 정수(32비트, 곱하지 않은 픽셀 값)입니다.

반대로 픽셀 그룹의 값을 변경하거나 설정하려면 setPixels() 메서드를 사용합니다. 이 메서드는 두 매개 변수(rect 및 inputByteArray)가 입력될 때까지 기다린 후 이 둘을 결합하여 픽셀 데이터(inputByteArray)의 사각형 영역(rect)을 출력합니다.

inputByteArray에서 데이터를 읽고 기록하면 배열의 각 픽셀에 대해 ByteArray.readUnsignedInt() 메서드가 호출됩니다. 어떤 이유로 인해 inputByteArray에 픽셀 데이터에 해당하는 전체 사각형이 포함되어 있지 않으면 해당 시점에서 메서드가 이미지 처리를 중단합니다.

픽셀 데이터 가져오고 설정하려면 바이트 배열이 32비트 알파, 빨강, 녹색, 파랑(ARGB) 픽셀 값이어야 합니다.

다음 예제에서는 getPixels() 및 setPixels() 메서드를 사용하여 픽셀 그룹을 한 BitmapData 객체에서 다른 BitmapData 객체로 복사합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false,
  0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false,
  0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

픽셀 수준 충돌 감지

`BitmapData.hitTest()` 메서드는 비트맵 데이터와 다른 객체 또는 점 사이에서 픽셀 수준의 충돌이 있었는지 여부를 감지합니다.

`BitmapData.hitTest()` 메서드는 다음 5개의 매개 변수를 사용합니다.

- `firstPoint`(점): 히트 테스트가 수행되는 첫 번째 BitmapData의 왼쪽 위 모서리의 픽셀 위치를 나타내는 매개 변수입니다.
- `firstAlphaThreshold`(단위): 해당 히트 테스트에 대해 불투명으로 간주되는 최고 알파 채널 값을 지정하는 매개 변수입니다.
- `secondObject`(객체): 영향을 받는 영역을 나타내는 매개 변수입니다. `secondObject` 객체는 Rectangle, Point, Bitmap 또는 BitmapData 객체일 수 있습니다. 또한 충돌 감지가 수행되는 히트 영역을 나타냅니다.
- `secondBitmapDataPoint`(점): 2차 BitmapData 객체의 픽셀 위치를 정의하는 데 사용되는 선택적 매개 변수로서, `secondObject`의 값이 BitmapData 객체일 경우에만 사용됩니다. 기본값은 `null`입니다.
- `secondAlphaThreshold`(단위): 2차 BitmapData 객체에서 불투명으로 간주되는 최고 알파 채널 값을 나타내는 선택적 매개 변수로서, 기본값은 1이며 `secondObject`의 값이 BitmapData 객체이고 두 BitmapData 객체가 모두 투명 객체일 때만 사용됩니다.

불투명 이미지에서 충돌 감지 작업을 수행할 경우 ActionScript에서는 이미지를 완전히 불투명한 사각형(또는 경계 상자)처럼 처리한다는 것을 염두에 두십시오. 또는 투명한 이미지에서 픽셀 수준의 히트 테스트를 수행할 경우 두 이미지 모두 투명해야 합니다. 이외에도 ActionScript는 알파 임계값 매개 변수를 사용하여 픽셀이 투명에서 불투명으로 바뀌는 지점을 결정합니다.

다음 예제에서는 세 개의 비트맵 이미지를 만든 다음 두 개의 서로 다른 충돌점(한 점에서는 false를, 다른 점에서는 true를 반환)을 사용하여 픽셀 충돌 여부를 확인합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

비트맵 데이터 복사

clone(), copyPixels(), copyChannel() 및 draw() 등과 같은 여러 메서드를 사용하여 한 이미지의 비트맵 데이터를 다른 이미지로 복사할 수 있습니다.

clone() 메서드는 이름에서도 알 수 있듯이 비트맵 데이터를 한 BitmapData 객체에서 다른 BitmapData 객체로 복제하거나 샘플링합니다. 이 메서드를 호출하면 BitmapData 객체를 복제한 원본 인스턴스를 똑같이 복제한 새 BitmapData 객체가 반환됩니다.

다음 예제에서는 주황색(부모) 정사각형을 복제한 후 원본 부모 정사각형 옆에 해당 복제본을 배치합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false,
  0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

copyPixels() 메서드는 BitmapData 객체 간에 픽셀을 복사하는 쉽고 빠른 방법입니다. 이 메서드는 소스 이미지의 사각형 스냅샷(sourceRect 매개 변수로 정의됨)을 가져와서 같은 크기의 다른 사각형 영역으로 복사합니다. 새로 "붙여넣은" 사각형의 위치는 destPoint 매개 변수 내에 정의됩니다.

copyChannel() 메서드는 소스 BitmapData 객체에서 미리 정의된 색상 채널 값(알파, 빨강, 녹색 또는 파랑)을 샘플링하여 대상 BitmapData 객체의 채널에 복사합니다. 이 메서드를 호출하더라도 대상 BitmapData 객체의 다른 채널은 영향을 받지 않습니다.

draw() 메서드는 소스 Sprite, 무비 클립 또는 다른 표시 객체의 그래픽 내용을 새 비트맵에 렌더링하거나 그립니다. matrix, colorTransform, blendMode 및 대상 clipRect 매개 변수를 사용하면 새 비트맵이 렌더링되는 방식을 수정할 수 있습니다. 이 메서드는 Flash Player 벡터 렌더러를 사용하여 데이터를 생성합니다.

draw()를 호출하면 여기서 설명한 것처럼 소스 객체(예: Sprite, 무비 클립, 기타 표시 객체)가 첫 번째 매개 변수로 전달됩니다.

```
myBitmap.draw(movieClip);
```

소스 객체가 처음 로드된 후에 색상 및 매트릭스 등의 변형이 적용된 경우, 이러한 변형은 새 객체로 복사되지 않습니다. 변형을 새 비트맵으로 복사하려면 원본 객체의 transform 속성 값을 새 BitmapData 객체를 사용하는 Bitmap 객체의 transform 속성에 복사해야 합니다.

노이즈 함수를 사용하여 텍스처 만들기

noise() 메서드 또는 perlinNoise() 메서드를 통해 비트맵에 노이즈 효과를 적용하여 비트맵 모양을 수정할 수 있습니다. 노이즈 효과는 튜닝되지 않은 텔레비전 화면에 나타나는 잡음에 비유할 수 있습니다.

비트맵에 노이즈 효과를 적용하려면 noise() 메서드를 사용하십시오. 이 메서드는 비트맵 이미지의 특정 영역에 있는 픽셀에 임의의 색상 값을 적용합니다.

이 메서드는 다음 5개의 매개 변수를 사용합니다.

- randomSeed(정수): 패턴을 결정하는 난수 초기값입니다. 이름과는 달리 이 숫자는 사실상 동일한 숫자가 전달될 경우 동일한 결과를 생성합니다. 따라서 의미 있는 임의의 결과를 얻으려면 Math.random() 메서드를 사용하여 이 매개 변수의 난수를 전달해야 합니다.
- low(단위): 각 픽셀(0 ~ 255)에 대해 생성되는 최저 값을 나타내는 매개 변수로서, 기본값은 0입니다. 이 값을 낮게 설정하면 노이즈 패턴이 어두워지고 높게 설정하면 노이즈 패턴이 밝아집니다.
- high(단위): 각 픽셀(0 ~ 255)에 대해 생성되는 최고 값을 나타내는 매개 변수로서, 기본값은 255입니다. 이 값을 낮게 설정하면 노이즈 패턴이 어두워지고 높게 설정하면 노이즈 패턴이 밝아집니다.
- channelOptions(단위): 노이즈 패턴이 적용될 비트맵 객체의 색상 채널을 지정하는 매개 변수로서, 네 가지 색상 채널 ARGB 값을 임의로 결합한 숫자가 될 수 있습니다. 기본값은 7입니다.
- grayScale(부울): true로 설정된 경우 randomSeed 값을 비트맵 픽셀에 적용하여 이미지의 모든 색상을 효과적으로 지웁니다. 알파 채널은 이 매개 변수의 영향을 받지 않습니다. 기본값은 false입니다.

다음 예제에서는 비트맵 이미지를 생성한 후 이 이미지에 파란색 노이즈 패턴을 적용합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

좀 더 유기적인 텍스처를 만들려면 perlinNoise() 메서드를 사용합니다. perlinNoise() 메서드는 연기, 구름, 물, 불 또는 폭발 등의 효과를 표현하는 데 적합한 사실적이고 유기적인 텍스처를 생성합니다.

perlinNoise() 메서드는 알고리즘을 통해 텍스처를 생성하기 때문에 비트맵 기반 텍스처보다 사용되는 메모리가 적습니다. 그러나 프로세스 사용에는 영향을 미치므로 Flash의 내용이 느리게 표시될 수 있으며 그 결과 특히 오래된 컴퓨터에서는 스크린이 프레임 속도보다 느린 속도로 다시 그려집니다. 이와 같은 문제는 주로 Perlin 노이즈 알고리즘을 처리하기 위해 부동 소수점 계산을 수행할 때 발생합니다.

이 메서드는 다음과 같은 9개의 매개 변수를 사용하며, 이 중 처음 6개는 필수 매개 변수입니다.

- baseX(숫자): 만들어진 패턴의 x(크기) 값을 결정합니다.
- baseY(숫자): 만들어진 패턴의 y(크기) 값을 결정합니다.
- numOctaves(단위): 이 노이즈를 생성하기 위해 결합할 옥타브, 즉 개별 노이즈 함수의 수입니다. 옥타브 수가 많을수록 보다 상세한 이미지가 만들어지지만 처리하는 데 시간이 오래 걸립니다.
- randomSeed(정수): 난수 초기값으로 noise() 함수에서 작동하는 것과 같은 방식으로 작동합니다. 따라서 의미 있는 임의의 결과를 얻으려면 Math.random() 메서드를 사용하여 이 매개 변수의 난수를 전달해야 합니다.
- stitch(부울): true로 설정된 경우 이 메서드는 비트맵 채우기 작업 시 연속 타일링 텍스처를 만들어내기 위해 이미지의 가장자리를 매끄럽게 하려고 시도합니다.
- fractalNoise(부울): 메서드에 의해 생성된 그래디언트의 가장자리와 관련된 매개 변수입니다. true로 설정된 경우 해당 효과의 가장자리를 매끄럽게 하는 프랙탈 노이즈를 생성합니다. false로 설정된 경우에는 난류가 생성됩니다. 난류가 포함된 이미지의 경우 그래디언트에서 불연속 선이 드러나기 때문에 불꽃과 파도 같은 시각적 효과를 만드는 데 적합합니다.
- channelOptions(단위): channelOptions 매개 변수는 noise() 메서드에서 작동하는 것과 같은 방식으로 작동합니다. 또한 노이즈 패턴이 적용되는 비트맵의 색상 채널을 지정합니다. 네 가지 색상 채널 ARGB 값을 임의로 결합한 숫자가 될 수 있습니다. 기본값은 7입니다.
- grayScale(부울): grayScale 매개 변수는 noise() 메서드에서 작동하는 것과 같은 방식으로 작동합니다. true로 설정된 경우 randomSeed 값을 비트맵 픽셀에 적용하여 이미지의 모든 색상을 효과적으로 지웁니다. 기본값은 false입니다.
- offsets(배열): 각 옥타브의 x 및 y 오프셋에 해당하는 점 배열입니다. 오프셋 값을 조작하면 이미지 레이어를 부드럽게 스크롤할 수 있습니다. 오프셋 배열의 각 점은 특정 옥타브 노이즈 함수에 영향을 미칩니다. 기본값은 null입니다.

다음 예제에서는 perlinNoise() 메서드를 호출하여 녹색 및 파란색 구름 효과를 생성하는 150 x 150픽셀의 BitmapData 객체를 만듭니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
```

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false,
  0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels,
  false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

비트맵 스크롤

사용자가 지도를 움직일 때마다(단 몇 개의 픽셀만 움직였을 경우에도) 화면을 업데이트해야 하는 주소 매핑 응용 프로그램을 개발했다고 가정합시다.

이러한 기능을 구현하는 방법 중 하나는 사용자가 지도를 움직일 때마다 업데이트된 맵 화면이 포함된 새 이미지를 다시 렌더링하는 것입니다. 또는 한 개의 커다란 이미지와 `scroll()` 메서드를 만드는 방법도 생각할 수 있습니다.

`scroll()` 메서드는 화면상에 있는 비트맵을 복사하여 `x`, `y` 매개 변수에서 지정한 새로운 오프셋 위치에 붙여 넣습니다. 비트맵의 일부가 안 보이는 위치에 있을 경우 이 메서드는 이미지가 이동한 것처럼 보이는 효과를 연출합니다. 또한 타이머 함수(또는 `enterFrame` 이벤트)와 함께 사용할 경우 이미지에 애니메이션 효과나 스크롤 효과를 줄 수 있습니다.

다음 예제에서는 앞서 설명한 Perlin 노이즈 예제를 활용하여 이미지의 3/4이 스테이지에 표시되지 않는 큰 비트맵 이미지를 생성합니다. 그런 다음 `scroll()` 메서드와 이미지를 대각선 아래쪽 방향으로 1픽셀씩 오프셋하는 `enterFrame` 이벤트 리스너를 적용합니다. 이 메서드는 해당 프레임에 진입할 때마다 호출되며 그 결과, 이미지를 아래로 스크롤할 때 스크린 밖의 이미지 부분을 스테이지로 렌더링합니다.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false,
  0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels,
  false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);
```

```
function scrollBitmap(event:Event):void
{
  myBitmapDataObject.scroll(1, 1);
}
```

예제: 오프스크린 비트맵을 사용하여 Sprite 애니메이션 적용

상당수의 Flash 게임은 한 스크린에서 동시에 이루어지는 수많은 이미지 애니메이션으로 구성됩니다. 이 비트맵 애니메이션 예제는 수백 개의 작은 비트맵 또는 Sprite를 스크린에 표시되지 않는 하나의 큰 비트맵에 그린 다음 비트맵을 한 개씩 스크린에 표시하여 애니메이션 속도를 현저히 빠르게 합니다. 이 예제에 대한 설명을 보고 소스 코드를 다운로드하려면 www.adobe.com/go/learn_fl_bitmaps_kr을 방문하십시오.

제 19 장

19 비디오를 사용한 작업

Flash 비디오는 인터넷에서 두각을 나타내고 있는 기술 중 하나입니다. 그러나 아래쪽에 진행률 막대와 몇 가지 제어 버튼이 있는 직사각형 스크린에 비디오를 표시하는 것은 Flash 응용 프로그램에서 비디오를 사용한 작업 중 하나일 뿐입니다. ActionScript를 통해 비디오 로드, 표시 및 재생을 세밀하게 조정하고 제어할 수 있습니다.

목차

비디오의 기초

비디오를 사용한 작업 소개

Adobe Flash Player의 중요한 기능 중 하나는 기타 시각적 내용(예: 이미지, 애니메이션, 텍스트)을 조작하는 것과 동일한 방법으로 ActionScript를 사용하여 비디오 정보를 표시하고 조작하는 것입니다.

Adobe Flash CS3 Professional에서 Flash 비디오(FLV) 파일을 만드는 경우 일반적인 재생 컨트롤을 포함한 비디오 스킨을 선택하는 옵션이 있습니다. 그러나 제공되는 옵션 외에도 다양한 기능을 활용할 수 있습니다. ActionScript를 사용하면 비디오 재생 로드, 표시 및 제어를 세밀하게 조절하여 자신만의 비디오 플레이어 스킨을 만들거나 사용자가 원하는 새로운 방식으로 비디오를 사용할 수 있습니다.

ActionScript에서 비디오를 사용한 작업을 수행하려면 다음과 같은 여러 클래스를 조합해서 사용해야 합니다.

- Video 클래스: 스테이지에 있는 실제 비디오 내용 상자가 Video 클래스의 인스턴스입니다. Video 클래스는 표시 객체이므로 위치 지정, 변형 적용, 필터 및 블렌드 모드 적용을 비롯하여 다른 표시 객체에 적용할 수 있는 동일한 기술로 조작할 수 있습니다.
- NetStream 클래스: ActionScript에서 제어하도록 비디오 파일을 로드할 때, 비디오 내용의 소스(이 경우에는 비디오 데이터 스트림)를 나타내는 데 NetStream 인스턴스가 사용됩니다. NetStream 인스턴스를 사용하려면 NetConnection 객체도 사용해야 합니다. NetConnection 객체는 비디오 파일에 대한 연결이며 비디오 데이터가 제공되는 터널과 같습니다.
- Camera 클래스: 사용자의 컴퓨터에 연결된 카메라에서 입력된 비디오 데이터를 사용하여 작업할 때 Camera 인스턴스는 비디오 내용의 소스 즉, 사용자의 카메라 및 비디오 데이터를 통해 입력된 내용을 나타냅니다.

외부 비디오를 로드할 때는 표준 웹 서버에서 파일을 로드하여 점진적인 다운로드 재생을 하거나, Adobe의 Macromedia® Flash® Media Server와 같은 특별 서버에 의해 전달된 비디오를 스트리밍할 수도 있습니다.

일반적인 비디오 작업

이 장에서는 사용자가 수행할 다음과 같은 비디오 관련 작업에 대해 설명합니다.

- 스크린에서 비디오 표시 및 제어
- 외부 FLV 파일 로드
- 비디오 파일의 메타데이터 및 큐 포인트 정보 처리
- 사용자의 카메라에서 비디오 입력 캡처 및 표시

중요한 개념 및 용어

- 큐 포인트: 비디오 파일에서 특정 순간에 삽입할 수 있는 표시자입니다. 예를 들어, 특정 지점을 찾기 위한 책갈피로 큐 포인트를 사용하거나 특정 순간과 관련된 추가 데이터를 제공하기 위해 큐 포인트를 사용할 수 있습니다.
- 인코딩: 특정 포맷의 비디오 데이터를 다른 비디오 데이터 포맷으로 변환하는 프로세스입니다. 예를 들어, 고해상도의 소스 비디오를 인터넷으로 전송하는 데 적합한 포맷으로 변환할 수 있습니다.
- 프레임: 비디오 정보의 한 세그먼트로, 각 프레임은 한 순간의 스냅샷을 나타내는 정지 영상과 같습니다. 프레임을 고속으로 연속 재생하면 움직이는 것처럼 보입니다.
- 키프레임: 프레임의 모든 정보가 포함된 비디오 프레임입니다. 키프레임 다음에 오는 다른 프레임에는 전체 프레임의 정보가 아닌 키프레임과 구분되는 정보만 포함됩니다.
- 메타데이터: 비디오 파일에 포함할 수 있으며 비디오가 로드되었을 때 검색할 수 있는 비디오 파일에 대한 정보입니다.
- 점진적 다운로드: 표준 웹 서버에서 비디오 파일을 전송하는 경우 비디오 데이터는 점진적 다운로드를 통해 로드됩니다. 즉, 비디오 정보가 순서대로 로드됩니다. 점진적 다운로드를 사용하면 전체 파일이 다운로드되기 전에 비디오 재생을 시작할 수 있다는 장점이 있지만 아직 로드되지 않은 비디오 부분으로 이동할 수는 없습니다.
- 스트리밍: "트루 스트리밍"이라고 하는 스트리밍 기술을 사용하여 특수 비디오 서버를 통해 인터넷으로 비디오를 전송하는 방법으로서 점진적 다운로드 대신 사용할 수 있습니다. 스트리밍을 사용하면 해당 컴퓨터에 전체 비디오가 한 번에 다운로드되지 않습니다. 다운로드 속도를 높이기 위해 컴퓨터에서는 언제나 전체 비디오 정보의 일부만을 필요로 합니다. 특수 서버에서 비디오 내용 전송을 제어하므로 비디오 액세스를 위해 다운로드될 때까지 기다릴 필요 없이 언제든지 비디오의 모든 부분에 액세스할 수 있습니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장은 ActionScript의 비디오 작업을 다루므로, 이 장 내의 코드 샘플 중 상당수는 비디오 객체(Flash 제작 도구에서 만들어 스테이지에 배치된 객체 또는 ActionScript를 사용하여 만든 객체) 작업과 관련이 있습니다. 샘플을 테스트하려면 코드로 비디오를 조작한 결과를 Flash Player에서 확인해야 합니다.

대부분의 예제 코드 샘플은 Video 객체를 명시적으로 생성하지 않고 조작합니다. 이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.

4. 필요한 경우 [라이브러리] 패널을 엽니다.
5. [라이브러리] 패널의 메뉴에서 [새 비디오]를 선택합니다.
6. [비디오 속성] 대화 상자에서 새 비디오 심볼 이름을 입력한 다음, [유형] 필드에서 [비디오(ActionScript 조절됨)]을 선택합니다. [확인]을 클릭하여 비디오 심볼을 만듭니다.
7. 만든 비디오 심볼을 [라이브러리] 패널에서 스테이지로 드래그합니다.
8. 비디오 인스턴스를 선택한 상태에서, 속성 관리자에서 인스턴스 이름을 지정합니다. 이 이름은 예제 코드 샘플에서 해당 비디오 인스턴스에 대해 사용한 이름과 일치해야 합니다. 예를 들어, 코드 샘플에서 `vid`라는 Video 객체를 조작하려는 경우 스테이지 인스턴스 이름도 `vid`로 지정해야 합니다.
9. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   코드 샘플에 지정된 대로 코드를 통해 비디오를 조작한 결과가 스크린에 표시됩니다.

이 장의 예제 코드 샘플 중 일부에는 해당 예제 코드뿐만 아니라 클래스 정의도 포함되어 있습니다. 이러한 샘플의 경우, 앞의 단계와 더불어 SWF를 테스트하려면 예제에 사용할 클래스를 만들어야 합니다. 예제 코드 샘플에 정의된 클래스를 만들려면:

1. 테스트에 사용할 FLA 파일을 저장해 두었는지 확인합니다.
2. 주 메뉴에서 [파일] > [새로 만들기]를 선택합니다.
3. [새 문서] 대화 상자의 [유형] 섹션에서 [ActionScript 파일]을 선택합니다. [확인]을 클릭하여 새 ActionScript 파일을 만듭니다.
4. 예제의 클래스 정의 코드를 ActionScript 문서에 복사합니다.
5. 주 메뉴에서 [파일] > [저장]을 선택합니다. 파일을 Flash 문서와 같은 디렉토리에 저장합니다. 파일 이름은 코드 샘플에 있는 클래스의 이름과 일치해야 합니다. 예를 들어, 코드 샘플에서 "VideoTest"라는 클래스 이름을 정의했으면 ActionScript 파일을 "VideoTest.as"로 저장합니다.
6. Flash 문서로 돌아갑니다.
7. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   스크린에 표시되는 예제 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

Flash 비디오(FLV) 포맷 이해

FLV 파일 포맷에는 Flash Player를 사용한 전송을 위한 인코딩된 오디오 및 비디오 데이터가 포함됩니다. 예를 들어, QuickTime 또는 Windows Media 비디오 파일이 있는 경우 Flash Video Encoder 또는 Sorenson™ Squeeze 등과 같은 인코더를 사용하여 해당 파일을 FLV 파일로 변환할 수 있습니다.

비디오를 Flash 제작 도구로 가져온 후 FLV 파일로 내보내는 방법으로 FLV 파일을 만들 수 있습니다. 지원되는 비디오 편집 응용 프로그램에서 FLV 내보내기 플러그인을 사용하여 FLV 파일을 내보낼 수 있습니다.

외부 FLV 파일을 사용하면 가져온 비디오를 사용할 때 지원되지 않는 다음과 같은 기능을 사용할 수 있습니다.

- 재생 속도를 늦추지 않고도 Flash 문서에서 긴 비디오 클립을 사용할 수 있습니다. 외부 FLV 파일은 캐시된 메모리를 사용하여 재생되는데, 이는 큰 파일이 여러 개의 작은 부분으로 저장되어 동적으로 액세스되므로 포함된 비디오 파일보다 메모리가 적게 사용됨을 의미합니다.
- 외부 FLV 파일의 프레임 속도를 해당 파일이 재생되는 Flash 문서의 프레임 속도와 다르게 지정할 수 있습니다. 예를 들어, Flash 문서의 프레임 속도를 30fps(초당 프레임 수)로 설정하고 비디오 프레임 속도를 21fps로 설정할 수 있습니다. 이렇게 설정하면 포함된 비디오를 사용할 때보다 비디오를 세부적으로 제어하여 비디오를 매끄럽게 재생할 수 있습니다. 또한 기존 Flash 내용을 변경하지 않고 다른 프레임 속도로 FLV 파일을 재생할 수 있습니다.
- 외부 FLV 파일을 사용할 경우에는 비디오 파일을 로드하는 동안 Flash 문서의 재생을 중단할 필요가 없습니다. 가져온 비디오 파일은 CD-ROM 드라이브에 액세스하는 등의 특정 기능을 실행하기 위해 문서 재생을 방해할 수 있습니다. FLV 파일은 Flash 문서와는 별개로 기능을 실행할 수 있으므로 문서 재생을 방해하지 않습니다.
- 외부 FLV 파일의 경우 이벤트 핸들러를 사용하여 비디오의 메타데이터에 액세스할 수 있으므로 비디오 내용에 캡션을 쉽게 추가할 수 있습니다.

팁 웹 서버로부터 FLV 파일을 로드하려면 웹 서버에서 파일 확장자와 MIME 유형을 등록해야 할 수 있습니다. 웹 서버 설명서를 확인하십시오. FLV 파일의 MIME 유형은 `video/x-flv` 입니다. 자세한 내용은 546페이지의 "서버에 호스트할 수 있도록 FLV 파일 구성"을 참조하십시오.

Video 클래스 이해

Video 클래스를 사용하면 라이브 스트리밍 비디오를 SWF 파일에 포함하지 않고도 응용 프로그램에서 표시할 수 있습니다. `Camera.getCamera()` 메서드를 사용하여 라이브 비디오를 캡처하고 재생할 수 있습니다. 또한 Video 클래스를 사용하여 HTTP를 통하거나 로컬 파일 시스템으로부터 FLV 파일을 재생할 수도 있습니다. 프로젝트에서 Video 클래스를 사용하는 방법에는 여러 가지가 있습니다.

- NetConnection 및 NetStream 클래스를 사용하여 동적으로 FLV를 로드하고 비디오를 Video 객체에 표시합니다.
- 사용자의 카메라에서 입력을 캡처합니다.
- FLVPlayback 구성 요소를 사용합니다.

예제	스테이지에 있는 Video 객체의 인스턴스는 Video 클래스의 인스턴스입니다.

Video 클래스가 flash.media 패키지 내에 있지만 flash.display.DisplayObject 클래스에서 상속되므로, 행렬 변환 및 필터 등의 표시 객체 기능도 모두 Video 인스턴스에 적용됩니다.

자세한 내용은 372페이지의 “표시 객체 조작”, 405페이지의 “기하 도형을 사용한 작업” 및 437페이지의 “표시 객체 필터링”을 참조하십시오.

비디오 파일 로드

NetStream 및 NetConnection 클래스를 사용한 비디오 로드는 여러 단계로 구성된 프로세스입니다.

1. 첫 번째 단계는 NetConnection 객체를 만드는 것입니다. NetConnection 클래스는 Adobe Flash Media Server 2 또는 Adobe Flex 등의 서버를 사용하지 않는 로컬 FLV 파일에 연결되어 있는 경우 `connect()` 메서드에 `null` 값을 전달하여 HTTP 주소나 로컬 드라이브에서 스트리밍 FLV 파일을 재생하도록 합니다.

   ```
   var nc:NetConnection = new NetConnection();
   nc.connect(null);
   ```

2. 두 번째 단계는 NetConnection 객체를 매개 변수로 사용하여 로드할 FLV 파일을 지정하는 NetStream 객체를 만드는 것입니다. 다음 코드는 NetStream 객체를 지정된 NetConnection 인스턴스와 연결하고 video.flv라는 FLV를 SWF 파일과 동일한 디렉토리에 로드합니다.

   ```
   var ns:NetStream = new NetStream(nc);
   ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
   ns.play("video.flv");
   function asyncErrorHandler(event:AsyncErrorEvent):void
   {
     // 오류를 무시합니다 .
   }
   ```

3. 세 번째 단계는 새 Video 객체를 만들어 Video 클래스의 `attachNetStream()` 메서드를 사용하여 이전 단계에서 만든 NetStream 객체를 연결하는 것입니다. 그런 후, 다음 코드처럼 `addChild()` 메서드를 사용하여 Video 객체를 표시 목록에 추가할 수 있습니다.

```
var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

위의 코드를 입력하면 Flash Player는 video.flv 비디오 파일을 SWF 파일과 동일한 디렉토리에 로드하려고 시도합니다.

참고 웹 서버로부터 FLV 파일을 로드하려면 웹 서버에서 파일 확장자와 MIME 유형을 등록해야 할 수 있습니다. 웹 서버 설명서를 확인하십시오. FLV 파일의 MIME 유형은 `video/x-flv`입니다. 자세한 내용은 546페이지의 "서버에 호스트할 수 있도록 FLV 파일 구성"을 참조하십시오.

비디오 재생 제어

NetStream 클래스는 비디오 재생 제어를 위한 네 가지 주요 메서드를 제공합니다.

`pause()`: 비디오 스트림의 재생을 일시 정지합니다. 비디오가 이미 일시 정지된 경우 이 메서드를 호출해도 아무 것도 수행되지 않습니다.

`resume()`: 일시 정지했던 비디오 스트림의 재생을 다시 시작합니다. 비디오가 이미 재생 중인 경우 이 메서드를 호출하면 아무 작업도 수행되지 않습니다.

`seek()`: 지정된 위치(스트림 시작 부분부터 초 단위의 오프셋)에 가장 가까운 키프레임을 찾습니다.

`togglePause()`: 스트림의 재생을 일시 정지하거나 다시 시작합니다.

예외 `stop()` 메서드가 없습니다. 스트림을 멈추려면 재생을 일시 정지한 후 비디오 스트림의 시작 부분을 찾아야 합니다.

예외 `play()` 메서드는 재생을 다시 시작하지 않으며 비디오 파일 로드에 사용됩니다.

다음 예제에서는 여러 버튼을 사용하여 비디오를 제어하는 방법을 설명합니다. 다음 예제를 실행하려면 새 문서를 만들고 작업 영역에 버튼 인스턴스를 네 개(`pauseBtn`, `playBtn`, `stopBtn`, `togglePauseBtn`) 추가합니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
  // 오류를 무시합니다 .
```

```
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
  ns.pause();
}
function playHandler(event:MouseEvent):void
{
  ns.resume();
}
function stopHandler(event:MouseEvent):void
{
  // 스트림을 일시 정지하고 재생 헤드를 스트림의 시작
  // 부분으로 다시 이동합니다 .
  ns.pause();
  ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
  ns.togglePause();
}
```

비디오가 재생되는 동안 pauseBtn 버튼 인스턴스를 클릭하면 비디오 파일이 일시 정지됩니다. 비디오가 이미 일시 정지된 경우에는 이 버튼을 클릭해도 아무 효과가 없습니다. 재생이 이미 일시 정지되어 있는 경우 playBtn 버튼 인스턴스를 클릭하면 비디오 재생이 다시 시작되지만, 비디오가 이미 재생 중인 경우에는 이 버튼을 클릭해도 아무 효과가 없습니다.

비디오 스트림의 끝 감지

비디오 스트림의 시작과 끝을 수신하려면 이벤트 리스너를 NetStream 인스턴스에 추가하여 netStatus 이벤트를 수신해야 합니다. 다음 코드에서는 비디오 재생 중 다양한 코드를 수신하는 방법을 설명합니다.

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
  trace(event.info.code)
}
```

이전 코드는 다음을 출력합니다.

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

특히 수신할 두 코드는 비디오 재생의 시작과 끝을 알리는 "NetStream.Play.Start"와 "NetStream.Play.Stop"입니다. 다음 코드에서는 switch 문을 사용하여 이러한 두 코드를 필터링하고 메시지를 추적합니다.

```
function statusHandler(event:NetStatusEvent):void
{
  switch (event.info.code)
  {
    case "NetStream.Play.Start":
      trace("Start [" + ns.time.toFixed(3) + " seconds]");
      break;
    case "NetStream.Play.Stop":
      trace("Stop [" + ns.time.toFixed(3) + " seconds]");
      break;
  }
}
```

사용자는 `netStatus` 이벤트(`NetStatusEvent.NET_STATUS`)를 수신하여 현재 비디오 재생 완료 시 목록에 있는 다음 비디오를 로드하는 비디오 플레이어를 만들 수 있습니다.

비디오 파일 스트리밍

Flash Media Server로부터 파일을 스트리밍하려는 경우 NetConnection 및 NetStream 클래스를 사용하여 원격 서버 인스턴스에 연결하고 지정된 스트림을 재생할 수 있습니다. RTMP(Real-Time Messaging Protocol) 서버를 지정하려면 `NetConnection.connect()` 메서드에 `null`이 아니라 원하는 RTMP URL(예: "rtmp://localhost/appName/appInstance")을 전달합니다. 지정된 Flash Media Server에서 특정 라이브 또는 녹화된 스트림을 재생하려면 `NetStream.publish()`에 의해 제작된 라이브 데이터를 식별하는 이름을, 녹화된 스트림을 재생하려면 재생 항목에 대해 기록된 파일 이름을 `NetStream.play()` 메서드에 전달합니다. 자세한 내용은 Flash Media Server 설명서를 참조하십시오.

큐 포인트 이해

모든 FLV 파일에 큐 포인트가 있는 것은 아닙니다. 기존 FLV 파일에 큐 포인트를 포함하기 위한 도구가 있지만 일반적으로 큐 포인트는 FLV 인코딩 중 FLV 파일에 포함됩니다.

다양한 유형의 큐 포인트를 Flash 비디오에 사용할 수 있습니다. ActionScript를 사용하여 FLV 파일 작성 시 FLV 파일에 포함하는 큐 포인트나 ActionScript로 만드는 큐 포인트와 상호 작용할 수 있습니다.

- 내비게이션 큐 포인트: FLV 파일을 인코딩할 때 FLV 스트림과 FLV 메타데이터 패킷에 내비게이션 큐 포인트를 포함합니다. 내비게이션 큐 포인트를 사용하면 사용자가 파일의 지정된 부분을 검색할 수 있습니다.
- 이벤트 큐 포인트: FLV 파일을 인코딩할 때 FLV 스트림과 FLV 메타데이터 패킷에 이벤트 큐 포인트를 포함합니다. FLV 재생 중 지정된 지점에서 트리거되는 이벤트를 처리하는 코드를 작성할 수 있습니다.
- ActionScript 큐 포인트: ActionScript 코드를 사용하여 만드는 외부 큐 포인트입니다. 비디오의 재생과 관련하여 이러한 큐 포인트를 트리거하는 코드를 작성할 수 있습니다. 이러한 큐 포인트는 비디오 플레이어에서 개별적으로 추적하므로 포함된 큐 포인트보다 최대 0.1초 정도 정확성이 떨어집니다.

내비게이션 큐 포인트는 지정된 큐 포인트 위치에 키프레임을 만들므로 코드를 사용하여 비디오 플레이어의 재생 헤드를 해당 위치로 이동할 수 있습니다. FLV 파일에 사용자가 검색할 수 있는 특정 지점을 설정할 수 있습니다. 예를 들어, 비디오에 여러 장이나 세그먼트가 있을 수 있으며 비디오 파일에 내비게이션 큐 포인트를 포함하여 비디오를 제어할 수 있습니다.

사용자의 큐 포인트 탐색이 가능한 응용 프로그램을 만들려면 ActionScript 큐 포인트를 사용하는 대신 파일을 인코딩할 때 큐 포인트를 만들거나 포함해야 합니다. 큐 포인트가 사용하기에 보다 정확하므로 FLV 파일에 큐 포인트를 포함해야 합니다. 큐 포인트를 사용한 FLV 파일 인코딩에 대한 자세한 내용은 *Flash 사용 설명서*의 "큐 포인트 포함"을 참조하십시오.

ActionScript를 작성하여 큐 포인트 매개 변수에 액세스할 수 있습니다. 큐 포인트 매개 변수는 `onCuePoint` 콜백 핸들러에서 수신된 이벤트 객체의 일부입니다.

비디오가 특정 큐 포인트에 도달했을 때 코드에서 특정 액션을 트리거하려면 `NetStream.onCuePoint` 이벤트 핸들러를 사용합니다. 자세한 내용은 529페이지의 "onCuePoint 및 onMetaData에 대한 콜백 메서드 작성"을 참조하십시오.

onCuePoint 및 onMetaData에 대한 콜백 메서드 작성

특정 큐 포인터에 도달하거나 플레이어에서 특정 메타데이터를 받을 때 응용 프로그램에서 액션을 트리거할 수 있습니다. 이러한 액션을 트리거하려면 onCuePoint 및 onMetaData 이벤트 핸들러를 사용합니다. 이러한 핸들러에 대한 콜백 메서드를 작성해야 합니다. 그렇지 않으면 Flash Player에서 오류가 발생할 수 있습니다. 예를 들어, 다음 코드에서는 SWF 문서와 같은 폴더에서 video.flv라는 FLV 파일을 재생합니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
  trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

앞의 코드는 video.flv라는 로컬 FLV 파일을 로드하며, 전달할 asyncError (AsyncErrorEvent.ASYNC_ERROR)를 수신합니다. 기본 비동기 코드에서 예외가 발생하면 이 이벤트가 전달됩니다. 이 경우 FLV에 메타데이터나 큐 포인트 정보가 있으면 이벤트가 전달되며 해당 리스너는 정의되어 있지 않습니다. 앞의 코드는 asyncError 이벤트를 처리하며, 사용자가 비디오 파일의 메타데이터나 큐 포인트 정보에 관심이 없는 경우 오류를 무시합니다. 메타데이터와 여러 큐 포인트가 포함된 FLV가 있는 경우 다음 정보가 추적됩니다.

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

이러한 오류는 NetStream 객체가 onMetaData 또는 onCuePoint 콜백 메서드를 찾지 못해서 발생합니다. 다음과 같은 여러 가지 방법으로 응용 프로그램 내에서 이러한 콜백 메서드를 정의할 수 있습니다.

- NetStream 객체의 client 속성을 Object로 설정
- 사용자 정의 클래스 만들기 및 콜백 메서드를 처리할 메서드 정의
- NetStream 클래스 확장 및 콜백 메서드를 처리할 메서드 추가
- NetStream 클래스 확장 및 동적 클래스로 만들기
- NetStream 객체의 client 속성을 this로 설정

NetStream 객체의 client 속성을 Object로 설정

client 속성을 Object 또는 NetStream 하위 클래스로 설정하여 onMetaData 및 onCuePoint 콜백 메서드를 다시 라우팅하거나 완전히 무시할 수 있습니다. 다음 예제는 asyncError 이벤트를 수신하지 않고 빈 Object를 사용하여 콜백 메서드를 무시할 수 있는 방법을 보여 줍니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

onMetaData 또는 onCuePoint 콜백 메서드 중 하나를 수신하려는 경우에는 다음 코드와 같이 해당 콜백 메서드를 처리할 메서드를 정의해야 합니다.

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
```

앞의 코드는 onMetaData 콜백 메서드를 수신하며 문자열을 추적하는 metaDataHandler() 메서드를 호출합니다. Flash Player에서 큐 포인트를 발견하는 경우에는 onCuePoint 콜백 메서드가 정의되지 않았더라도 오류가 생성되지 않습니다.

사용자 정의 클래스 만들기 및 콜백 메서드를 처리할 메서드 정의

다음 코드는 NetStream 객체의 client 속성을 콜백 메서드에 대한 핸들러를 정의하는 사용자 정의 클래스인 CustomClient로 설정합니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

CustomClient 클래스의 주요 기능은 다음과 같습니다.

```
package
{
  public class CustomClient
  {
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
  }
}
```

CustomClient 클래스는 onMetaData 콜백 핸들러에 대한 핸들러를 정의합니다. 큐 포인트가 발견되고 onCuePoint 콜백 핸들러가 호출되면 "flash.net.NetStream이 onCuePoint 콜백을 호출하지 못했습니다."라는 asyncError 이벤트(AsyncErrorEvent.ASYNC_ERROR)가 전달됩니다. 이러한 오류를 방지하려면 CustomClient 클래스에 onCuePoint 콜백 메서드를 정의하거나 asyncError 이벤트에 대한 이벤트 핸들러를 정의해야 합니다.

NetStream 클래스 확장 및 콜백 메서드를 처리할 메서드 추가

다음 코드는 CustomNetStream 클래스의 인스턴스를 만듭니다(이 클래스는 이후 코드 샘플에서 정의됨).

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

다음 코드 샘플은 NetStream 클래스를 확장하고 필수 NetConnection 객체 생성을 처리하며 onMetaData 및 onCuePoint 콜백 핸들러 메서드를 처리하는 CustomNetStream 클래스를 정의합니다.

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public function CustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
```

```
    }
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
    public function onCuePoint(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

CustomNetStream 클래스의 `onMetaData()` 및 `onCuePoint()` 메서드의 이름을 변경하려면 다음 코드를 사용할 수 있습니다.

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public var onMetaData:Function;
    public var onCuePoint:Function;
    public function CustomNetStream()
    {
      onMetaData = metaDataHandler;
      onCuePoint = cuePointHandler;
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
    private function metaDataHandler(infoObject:Object):void
    {
      trace("metadata");
    }
    private function cuePointHandler(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

NetStream 클래스 확장 및 동적 클래스로 만들기

NetStream 클래스를 확장하고 그 하위 클래스를 동적으로 만들어 onCuePoint 및 onMetaData 콜백 핸들러가 동적으로 추가될 수 있도록 할 수 있습니다. 다음 샘플에서는 이 방법을 설명합니다.

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

DynamicCustomNetStream 클래스의 주요 기능은 다음과 같습니다.

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public dynamic class DynamicCustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public function DynamicCustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
  }
}
```

onMetaData 및 onCuePoint 콜백 핸들러에 대한 핸들러가 없어도 DynamicCustomNetStream 클래스가 동적이므로 오류가 반환되지 않습니다. onMetaData 및 onCuePoint 콜백 핸들러에 대한 메서드를 정의하려면 다음 코드를 사용할 수 있습니다.

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
  trace("cue point");
}
```

NetStream 객체의 client 속성을 this로 설정

client 속성을 this로 설정하면 Flash Player는 현재 범위에서 onMetaData() 및 onCuePoint() 메서드를 찾습니다. 다음 예제에서 이를 확인할 수 있습니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

onMetaData 또는 onCuePoint 콜백 핸들러가 호출된 상태에서 콜백을 처리할 메서드가 없는 경우 오류가 생성되지 않습니다. 이러한 콜백 핸들러를 처리하려면 다음 코드와 같이 코드에 onMetaData() 및 onCuePoint() 메서드를 생성합니다.

```
function onMetaData(infoObject:Object):void
{
  trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
  trace("cue point");
}
```

큐 포인트 사용

다음 예제는 간단한 for..in 루프를 사용하여 onCuePoint 콜백 핸들러의 infoObject 매개 변수에서 각 속성을 반복하고, 큐 포인트 데이터 수신 시 메시지를 추적합니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
```

```
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

다음이 출력됩니다.

```
parameters:
name: point1
time: 0.418
type: navigation
```

이 코드에서는 콜백 메서드가 호출되는 객체를 설정하는 여러 방법 중 하나를 사용했습니다. 이외에 사용 가능한 다른 방법에 대한 자세한 내용은 onCuePoint 및 onMetaData에 대한 콜백 메서드 작성을 참조하십시오.

비디오 메타데이터 사용

onMetaData 콜백 핸들러를 사용하여 FLV 파일에서 메타데이터 정보를 볼 수 있습니다. 메타데이터에는 지속 시간, 폭, 높이 및 프레임 속도와 같은 FLV 파일에 대한 정보가 포함됩니다. FLV 파일에 추가되는 메타데이터 정보는 FLV 파일을 인코딩하는 데 사용하는 소프트웨어 또는 메타데이터 정보를 추가하는 데 사용하는 소프트웨어에 따라 다릅니다.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

사용자 FLV 파일에 큐 포인트 및 오디오가 포함되어 있는 경우, 이전의 코드에서 생성되는 코드는 다음과 유사합니다.

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
```

```
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

참고 **비디오에 오디오가 포함되지 않은 경우에는 인코딩 중에 메타데이터에 추가되는 오디오 정보가 없으므로 오디오 관련 메타데이터 정보(`audiodatarate` 등)가 `undefined`를 반환합니다.**

위의 코드에서는 큐 포인트 정보가 표시되지 않았습니다. 큐 포인트 메타데이터를 표시하려는 경우 Object에서 항목을 반복적으로 표시하는 다음과 같은 함수를 사용할 수 있습니다.

```
function traceObject(obj:Object, indent:uint = 0):void
{
  var indentString:String = "";
  var i:uint;
  var prop:String;
  var val:*;
  for (i = 0; i < indent; i++)
  {
    indentString += "\t";
  }
  for (prop in obj)
  {
    val = obj[prop];
    if (typeof(val) == "object")
    {
      trace(indentString + " " + j + ": [Object]");
      traceObject(val, indent + 1);
    }
    else
    {
      trace(indentString + " " + prop + ": " + val);
    }
  }
}
```

앞의 코드 예제를 사용하여 `onMetaData()` 메서드의 `infoObject` 매개 변수를 추적하면 다음이 출력됩니다.

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
```

```
height: 213
framerate: 15
cuePoints: [Object]
  0: [Object]
    parameters: [Object]
      lights: beginning
    name: point1
    time: 0.418
    type: navigation
  1: [Object]
    parameters: [Object]
      lights: middle
    name: point2
    time: 7.748
    type: navigation
  2: [Object]
    parameters: [Object]
      lights: end
    name: point3
    time: 16.02
    type: navigation
```

onMetaData에 대한 Info 객체

다음 표는 비디오 메타데이터로 가능한 값을 보여 줍니다.

매개 변수	설명
audiocodecid	사용된 오디오 코덱(코딩/디코딩 기술)을 나타내는 숫자입니다.
audiodatarate	오디오가 인코딩된 속도(초당 킬로바이트)를 나타내는 숫자입니다.
audiodelay	FLV 파일에서 원본 FLV 파일의 "time 0"이 있는 시간을 나타내는 숫자입니다. 오디오를 정확하게 동기화하기 위해서는 비디오 내용이 조금 지연되어야 합니다.
canSeekToEnd	부울 값으로서, FLV 파일이 점진적 다운로드 무비 클립의 끝까지 검색할 수 있도록 마지막 프레임의 키프레임으로 인코딩된 경우 `true`입니다. FLV 파일이 마지막 프레임의 키프레임으로 인코딩되지 않은 경우에는 `false`입니다.

매개 변수	설명
cuePoints	객체 배열이며, 배열의 각 객체는 FLV 파일에 포함된 각각의 큐 포인트에 해당합니다. FLV 파일에 큐 포인트가 없으면 이 값이 정의되지 않습니다. 각 객체에는 다음과 같은 속성이 있습니다. ■ `type`: 큐 포인트의 유형을 "navigation" 또는 "event" 중 하나로 지정하는 문자열입니다. ■ `name`: 큐 포인트의 이름을 나타내는 문자열입니다. ■ `time`: 큐 포인트의 시간을 소수 세 자리(밀리초)까지 정밀하게 초 단위로 나타내는 숫자입니다. ■ `parameters`: 큐 포인트를 만들 때 사용자가 지정한 이름/값 쌍을 포함하는 객체이며, 선택 사항입니다.
duration	FLV 파일의 지속 시간을 초 단위로 지정하는 숫자입니다.
framerate	FLV 파일의 프레임 속도를 나타내는 숫자입니다.
height	FLV 파일의 높이를 나타내는 숫자이며 픽셀 단위입니다.
videocodecid	비디오 인코딩에 사용된 코덱 버전을 나타내는 숫자입니다.
videodatarate	FLV 파일의 비디오 데이터 속도를 나타내는 숫자입니다.
width	FLV 파일의 폭을 나타내는 숫자이며 픽셀 단위입니다.

다음 표는 `videocodecid` 매개 변수로 가능한 값을 보여 줍니다.

videocodecid	코덱 이름
2	Sorenson H.263
3	스크린 비디오(SWF 7 이상에서만 사용 가능)
4	VP6(SWF 8 이상에서만 사용 가능)
5	알파 채널을 사용하는 VP6 비디오(SWF 8 이상에서만 사용 가능)

다음 표는 `audiocodecid` 매개 변수로 가능한 값을 보여 줍니다.

audiocodecid	코덱 이름
0	압축되지 않음
1	ADPCM
2	mp3

audiocodecid	코덱 이름
5	Nellymoser 8kHz 모노
6	Nellymoser

카메라 입력 캡처

ActionScript를 사용하여 표시하고 조작할 수 있는 비디오 데이터의 소스로 외부 비디오 파일뿐만 아니라 사용자의 컴퓨터에 연결된 카메라를 사용할 수 있습니다. Camera 클래스는 컴퓨터 카메라 사용을 위해 ActionScript에 내장된 메커니즘입니다.

Camera 클래스 이해

Camera 객체를 사용하면 사용자의 로컬 카메라에 연결하고 비디오를 로컬로 사용자에게 다시 브로드캐스팅하거나 원격으로 Flash Media Server 등의 서버에 브로드캐스팅할 수 있습니다.

Camera 클래스를 사용하여 사용자 카메라에 대한 다음과 같은 정보에 액세스할 수 있습니다.

- 사용자의 컴퓨터에 설치된 카메라 중 Flash Player에서 사용할 수 있는 카메라
- 카메라 설치 여부
- 사용자의 카메라에 대한 Flash Player의 액세스 허용 여부
- 현재 활성화된 카메라
- 캡처 중인 비디오의 폭 및 높이

Camera 클래스에는 Camera 객체를 사용하는 데 유용한 몇 가지 메서드와 속성이 있습니다. 예를 들어 정적 `Camera.names` 속성에는 사용자 컴퓨터에 현재 설치되어 있는 카메라 이름 배열이 포함됩니다. `name` 속성을 사용하여 현재 활성화된 카메라의 이름을 표시할 수도 있습니다.

스크린에 카메라 내용 표시

카메라에 연결하는 데 필요한 코드는 NetConnection 및 NetStream 클래스를 사용하여 FLV를 로드할 때보다 더 적을 수 있습니다. 그러나 카메라에 액세스할 수 있게 Flash Player를 카메라에 연결하려면 사용자가 이를 허용해야 하므로 Camera 클래스가 복잡해질 수 있습니다.

다음 코드에서는 Camera 클래스로 사용자의 로컬 카메라에 연결하는 방법을 설명합니다.

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
```

```
vid.attachCamera(cam);
addChild(vid);
```

예외	Camera 클래스에는 생성자 메서드가 없습니다. 새 Camera 인스턴스를 만들려면 정적 `Camera.getCamera()` 메서드를 사용합니다.

카메라 응용 프로그램 설계

사용자의 카메라에 연결되는 응용 프로그램을 작성할 때 다음 사항을 고려해야 합니다.

- 사용자가 카메라를 현재 설치해 둔 상태인지 확인합니다.
- 사용자가 Flash Player에서 카메라에 액세스할 수 있도록 명시적으로 허용했는지 확인합니다. 보안을 위하여 Flash Player가 [Flash Player 설정] 대화 상자를 표시하여 사용자가 카메라 액세스를 허용하거나 거부하도록 합니다. 이 대화 상자는 Flash Player에서 사용자의 허용 없이 카메라에 연결하여 비디오 스트림을 브로드캐스팅할 수 없게 합니다. 사용자가 [허용]을 클릭하면 응용 프로그램에서 사용자의 카메라에 연결할 수 있으며, [거부]를 클릭하면 응용 프로그램에서 사용자의 카메라에 액세스할 수 없습니다. 응용 프로그램에서 두 경우를 모두 적절하게 처리해야 합니다.

사용자의 카메라에 연결

사용자의 카메라에 연결하는 첫 번째 단계는 Camera 유형의 변수를 만든 다음 이 변수를 정적 `Camera.getCamera()` 메서드의 반환 값으로 초기화하여 새 Camera 인스턴스를 만드는 것입니다.

다음 단계는 새 Video 객체를 만들고 이 객체에 Camera 객체를 연결하는 것입니다.

세 번째 단계는 표시 목록에 Video 객체를 추가하는 것입니다. Camera 클래스는 DisplayObject 클래스를 확장하지 않아 표시 목록에 직접 추가할 수 없으므로 2단계와 3단계를 수행해야 합니다. 카메라에 캡처된 비디오를 표시하려면 새 Video 객체를 만들고 `attachCamera()` 메서드를 호출합니다.

다음 코드에서는 이러한 세 단계를 보여 줍니다.

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

사용자에게 설치된 카메라가 없는 경우 Flash Player는 아무 것도 표시하지 않습니다.

실제로 응용 프로그램에 사용할 때는 몇 가지 단계를 추가로 수행해야 합니다. 자세한 내용은 카메라 설치 여부 확인 및 카메라 액세스 허용 여부 확인을 참조하십시오.

카메라 설치 여부 확인

Camera 인스턴스에서 메서드나 속성을 사용하기 전에 사용자가 카메라를 설치했는지 확인해야 합니다. 다음 두 가지 방법으로 카메라 설치 여부를 확인할 수 있습니다.

- 사용할 수 있는 카메라 이름 배열이 포함된 정적 Camera.names 속성을 확인합니다. 대부분의 사용자는 한 번에 두 대 이상의 카메라를 설치하지 않으므로 일반적으로 이 배열에는 하나 이하의 문자열이 포함됩니다. 다음 코드는 Camera.names 속성을 점검하여 사용 가능한 카메라가 있는지 여부를 확인하는 방법을 보여 줍니다.

```
if (Camera.names.length > 0)
{
  trace("User has no cameras installed.");
}
else
{
  var cam:Camera = Camera.getCamera(); // 기본 카메라를 가져옵니다 .
}
```

- 정적 Camera.getCamera() 메서드가 반환하는 값을 확인합니다. 사용 가능하거나 설치되어 있는 카메라가 없는 경우 이 메서드는 null을 반환하고, 카메라가 있는 경우에는 Camera 객체를 반환합니다. 다음 코드는 Camera.getCamera() 메서드를 점검하여 사용 가능한 카메라가 있는지 여부를 확인하는 방법을 보여 줍니다.

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
  trace("User has no cameras installed.");
}
else
{
  trace("User has at least 1 camera installed.");
}
```

Camera 클래스는 DisplayObject 클래스를 확장하지 않으므로 addChild() 메서드를 사용하여 표시 목록에 직접 추가할 수 없습니다. 카메라에 캡처된 비디오를 표시하려면 새 Video 객체를 만들고 Video 인스턴스에 대해 attachCamera() 메서드를 호출해야 합니다.

다음 코드는 카메라가 있는 경우 카메라를 연결하는 방법을 보여 줍니다. 카메라가 없는 경우 Flash Player에 아무 것도 표시되지 않습니다.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
```

카메라 액세스 허용 여부 확인

사용자가 Flash Player에서 카메라에 액세스할 수 있도록 명시적으로 허용해야 카메라 출력을 표시할 수 있습니다. attachCamera() 메서드가 호출되면 Flash Player가 [Flash Player 설정] 대화 상자를 표시하여 사용자에게 Flash Player에서 카메라 및 마이크에 액세스하도록 허용할 것인지 여부를 선택하도록 요청합니다. 사용자가 [허용] 버튼을 클릭하면 스테이지의 Video 인스턴스에 카메라의 출력이 표시됩니다. 사용자가 [거부] 버튼을 클릭하면 Flash Player에서 카메라에 연결할 수 없으며 Video 객체가 아무 것도 표시하지 않습니다.

사용자가 카메라를 설치하지 않은 경우 Flash Player에 아무 것도 표시되지 않습니다. 사용자가 카메라를 설치한 경우에는 카메라에 대한 Flash Player의 액세스를 허용할지 또는 거부할지 묻는 [Flash Player 설정] 대화 상자가 사용자에게 표시됩니다. 사용자가 카메라에 대한 액세스를 허용하면 사용자에게 비디오가 다시 표시되고, 거부하면 아무 것도 표시되지 않습니다.

사용자가 카메라에 대한 액세스를 허용했는지 여부를 확인하려면 다음 코드와 같이 카메라의 status 이벤트(StatusEvent.STATUS)를 수신할 수 있습니다.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
  // 사용자가 Flash Player 설정 대화 상자에서 "허용" 또는 "거부" 버튼을
  // 클릭하면 이 이벤트가 전달됩니다.
  trace(event.code); // "Camera.Muted" 또는 "Camera.Unmuted"
}
```

사용자가 [허용] 또는 [거부]를 클릭하면 곧바로 statusHandler() 함수가 호출됩니다. 다음 두 메서드 중 하나를 사용하여 사용자가 클릭한 버튼을 확인할 수 있습니다.

- statusHandler() 함수의 event 매개 변수에는 "Camera.Muted" 또는 "Camera.Unmuted" 문자열이 들어 있는 코드 속성이 포함되어 있습니다. 값이 "Camera.Muted"이면 사용자가 [거부] 버튼을 클릭한 것이므로 Flash Player에서 카메라에 액세스할 수 없습니다. 다음 코드에서 이를 확인할 수 있습니다.

```
function statusHandler(event:StatusEvent):void
{
  switch (event.code)
  {
    case "Camera.Muted":
      trace("User clicked Deny.");
      break;
    case "Camera.Unmuted":
      trace("User clicked Accept.");
```

```
            break;
        }
    }
```

- Camera 클래스에는 `muted`라는 읽기 전용 속성이 포함되어 있으며 이 속성은 사용자가 카메라에 대한 액세스를 거부했는지(`true`) 아니면 허용했는지(`false`)를 Flash Player [개인 정보] 패널에 지정합니다. 다음 코드에서 이를 확인할 수 있습니다.

```
function statusHandler(event:StatusEvent):void
{
  if (cam.muted)
  {
    trace("User clicked Deny.");
  }
  else
  {
    trace("User clicked Accept.");
  }
}
```

전달할 상태 이벤트를 확인하여 카메라 액세스에 대한 사용자 허용 또는 거부를 처리하고 이에 맞게 정리하는 코드를 작성할 수 있습니다. 예를 들어, 사용자가 [거부] 버튼을 클릭할 경우 비디오 채팅에 참여하려면 [허용]을 클릭해야 한다는 메시지를 사용자에게 표시하거나, 표시 목록의 Video 객체를 삭제하여 사용 가능한 시스템 리소스를 늘릴 수 있습니다.

비디오 품질 최대화

기본적으로 Video 클래스의 새 인스턴스 크기는 폭 320픽셀 x 높이 240픽셀입니다. 비디오 품질을 최대화하려면 항상 Video 객체 크기를 Camera 객체에서 반환하는 비디오 크기에 맞춰야 합니다. Camera 클래스의 `width` 속성 및 `height` 속성을 사용하여 Camera 객체의 폭과 높이를 구할 수 있습니다. 그런 다음, 다음 코드 예제와 같이 Video 객체의 `width` 속성 및 `height` 속성을 Camera 객체 크기에 일치하도록 설정하거나, 카메라의 폭과 높이를 Video 클래스의 생성자 메서드에 전달할 수 있습니다.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video(cam.width, cam.height);
  vid.attachCamera(cam);
  addChild(vid);
}
```

`getCamera()` 메서드는 Camera 객체에 대한 참조를 반환하므로(또는 사용 가능한 카메라가 없는 경우 `null`을 반환), 사용자가 카메라에 대한 액세스를 거부한 경우에도 카메라의 메서드 및 속성에 액세스할 수 있습니다. 그러면 카메라의 기본 높이와 폭을 사용하여 비디오 인스턴스의 크기를 설정할 수 있습니다.

```
var vid:Video;
var cam:Camera = Camera.getCamera();
```

```
if (cam == null)
{
  trace("Unable to locate available cameras.");
}
else
{
  trace("Found camera: " + cam.name);
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
  if (cam.muted)
  {
    trace("Unable to connect to active camera.");
  }
  else
  {
    // 카메라 설정에 맞게 Video 객체의 크기를 조절하고
    // 표시 목록에 비디오를 추가합니다 .
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
  }
  // 상태 이벤트 리스너를 제거합니다 .
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

재생 조건 모니터링

Camera 클래스에는 Camera 객체의 현재 상태를 모니터링할 수 있는 몇 가지 속성이 있습니다. 예를 들어, 다음 코드에서는 Timer 객체와 표시 목록의 텍스트 필드 인스턴스를 사용하여 카메라의 여러 속성을 표시합니다.

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
```

```
function statusHandler(event:StatusEvent):void
{
  if (!cam.muted)
  {
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
    t.start();
  }
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
  tf.text = "";
  tf.appendText("activityLevel: " + cam.activityLevel + "\n");
  tf.appendText("bandwidth: " + cam.bandwidth + "\n");
  tf.appendText("currentFPS: " + cam.currentFPS + "\n");
  tf.appendText("fps: " + cam.fps + "\n");
  tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
  tf.appendText("loopback: " + cam.loopback + "\n");
  tf.appendText("motionLevel: " + cam.motionLevel + "\n");
  tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
  tf.appendText("quality: " + cam.quality + "\n");
}
```

매 1/10초(100밀리초)마다 Timer 객체의 `timer` 이벤트가 전달되고 `timerHandler()` 함수가 표시 목록의 텍스트 필드를 업데이트합니다.

서버에 비디오 보내기

Video 또는 Camera 객체가 포함된 보다 복잡한 응용 프로그램을 제작하려는 사용자를 위해 Flash Media Server는 미디어 스트리밍 기능과 미디어 응용 프로그램을 제작하여 더 많은 사용자에게 제공할 수 있는 개발 환경을 모두 제공합니다. 이를 통해 개발자는 VOD(주문형 비디오), 라이브 웹이벤트 방송, mp3 스트리밍, 비디오 블로그, 비디오 메시징, 멀티미디어 채팅 환경 등의 응용 프로그램을 제작할 수 있습니다. 자세한 내용은 http://livedocs.macromedia.com/fms/2/docs_kr에서 온라인으로 제공되는 Flash Media Server 설명서를 참조하십시오.

고급 항목

다음 항목에서는 비디오를 사용한 작업과 관련된 몇 가지 특수한 문제를 설명합니다.

Flash Player와 인코딩된 FLV 파일의 호환성

Flash Player 7에서는 Sorenson™ Spark™ 비디오 코덱을 사용하여 인코딩된 FLV 파일을 지원합니다. 또한 Flash Player 8에서는 Flash Professional 8에서 Sorenson Spark 또는 On2 VP6 인코더를 사용하여 인코딩된 FLV 파일을 지원합니다. On2 VP6 비디오 코덱은 알파 채널을 지원합니다. Flash Player 버전마다 다른 방식으로 FLV를 지원합니다. 자세한 내용은 다음 표를 참조하십시오.

코덱	SWF 파일 버전 (제작 버전)	재생에 필요한 Flash Player 버전
Sorenson Spark	6	6, 7 또는 8
	7	7, 8
On2 VP6	6	8*
	7	8
	8	8

* SWF 파일에서 FLV 파일을 로드하는 경우 사용자가 Flash Player 8을 사용하여 SWF 파일을 재생하는 한 Flash Player 8용으로 SWF 파일을 다시 제작할 필요 없이 On2 VP6 비디오를 사용할 수 있습니다. Flash Player 8에서만 On2 VP6 비디오의 제작과 재생을 지원합니다.

서버에 호스트할 수 있도록 FLV 파일 구성

FLV 파일을 사용하여 작업하는 경우 FLV 파일 포맷을 사용하도록 서버를 구성해야 합니다. MIME(Multipurpose Internet Mail Extensions)는 인터넷 연결로 ASCII가 아닌 파일을 보낼 수 있게 하는 표준화된 데이터 사양입니다. 웹 브라우저 및 전자 메일 클라이언트는 다양한 MIME 형식을 해석하도록 구성되어 있으므로 비디오, 오디오, 그래픽 및 서식 있는 텍스트를 보내고 받을 수 있습니다. 웹 서버로부터 FLV 파일을 로드하려는 경우 웹 서버에 파일 확장자와 MIME 유형을 등록해야 할 수 있습니다. 웹 서버 설명서에서 이를 확인하십시오. FLV 파일의 MIME 유형은 `video/x-flv`입니다. FLV 파일 유형에 대한 자세한 내용은 다음과 같습니다.

- Mime 유형: video/x-flv
- 파일 확장자: .flv
- 필수 매개 변수: 없음
- 선택적 매개 변수: 없음

- 인코딩 관련 고려 사항: FLV 파일은 이진 파일입니다. 따라서 일부 응용 프로그램의 경우 application/octet-stream 하위 유형을 설정해야 할 수 있습니다.
- 보안 문제: 없음
- 제작 사양: www.adobe.com/go/flashfileformat_kr

Microsoft IIS(인터넷 정보 서비스) 6.0 웹 서버에서는 이전 버전과 다른 방식으로 스트리밍 미디어를 처리합니다. 이전 버전의 IIS에서는 Flash 비디오를 스트리밍하기 위해 아무 것도 수정할 필요가 없었지만, IIS 6.0의 경우 Windows 2003과 함께 제공되는 기본 웹 서버에서 FLV 파일이 스트리밍 미디어임을 인식하려면 MIME 유형이 필요합니다.

외부 FLV 파일을 스트리밍하는 SWF 파일이 Microsoft Windows Server® 2003에 있고 브라우저에서 표시되는 경우 SWF 파일이 올바르게 재생되지만 FLV 비디오는 스트리밍되지 않습니다. 이 문제는 이전 버전 Flash 제작 도구인 Adobe Macromedia Flash Video Kit for Dreamweaver MX 2004를 사용하여 작성하는 파일을 비롯하여 Windows Server 2003에 있는 모든 FLV 파일에 영향을 미칩니다. 이러한 파일을 다른 운영 체제에서 테스트하면 올바르게 작동됩니다.

FLV 비디오를 스트리밍하도록 Microsoft Windows 2003 및 Microsoft IIS Server 6.0을 구성하는 방법에 대한 자세한 내용은 www.adobe.com/go/tn_19439_kr을 참조하십시오.

Macintosh에서 논리적 FLV 파일 대상 지정

상대 슬래시(/)를 사용하는 경로를 사용하여 Apple® Macintosh® 컴퓨터의 비 시스템 드라이브에서 논리적 FLV를 재생하려 할 경우 비디오가 재생되지 않습니다. 비 시스템 드라이브에는 CD-ROM, 파티션된 하드 디스크, 이동식 저장소 미디어 및 연결된 저장소 장치 등이 포함됩니다.

예외	이러한 오류는 Flash Player의 제한이 아니라 운영 체제의 제한 때문에 발생합니다.

Macintosh의 비 시스템 드라이브로부터 FLV 파일을 재생하려면 슬래시 기반 표기법(/)이 아닌 콜론 기반 표기법(:)을 사용하여 파일을 절대 경로로 참조해야 합니다. 다음 목록에서는 두 가지 표기법의 차이점을 보여 줍니다.

- 슬래시 기반 표기: myDrive/myFolder/myFLV.flv
- 콜론 기반 표기: (Mac OS®) myDrive:myFolder:myFLV.flv

Macintosh 재생에 사용할 CD-ROM용 프로젝터 파일을 만들 수도 있습니다. Mac OS CD-ROM 및 FLV 파일에 대한 최신 정보는 www.adobe.com/go/3121b301_kr을 참조하십시오.

예제: 비디오 주크박스

다음 예제에서는 순서대로 재생할 비디오 목록을 동적으로 로드하는 간단한 비디오 주크박스를 만듭니다. 이를 통해 사용자가 여러 비디오 설명서를 찾아볼 수 있는 응용 프로그램이나 사용자가 요청한 비디오를 전달하기 전에 재생할 광고를 지정하는 응용 프로그램을 만들 수 있습니다. 이 예제에서는 ActionScript 3.0의 다음 기능을 설명합니다.

- 비디오 파일의 재생 진행률에 따라 재생 헤드 업데이트
- 비디오 파일의 메타데이터 수신 및 구문 분석
- 넷스트림에서 특정 코덱 처리
- 동적으로 로드된 FLV 로드, 재생, 일시 정지 및 중지
- 넷스트림의 메타데이터를 기반으로 표시 목록의 Video 객체 크기 조절

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. 비디오 주크박스 응용 프로그램 파일은 Samples/VideoJukebox 폴더에 있습니다. 이 응용 프로그램은 다음과 같이 구성되어 있습니다.

파일	설명
VideoJukebox.as	응용 프로그램의 기본 기능을 제공하는 클래스입니다.
VideoJukebox.fla	Flash용 기본 응용 프로그램 파일입니다.
playlist.xml	비디오 주크박스로 로드할 비디오 파일이 나열된 파일입니다.

외부 비디오 재생 목록 파일 로드

외부 playlist.xml 파일은 로드할 비디오와 비디오 재생 순서를 지정합니다. XML 파일을 로드하려면 다음 코드와 같이 URLLoader 객체와 URLRequest 객체를 사용해야 합니다.

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

이 코드는 VideoJukebox 클래스의 생성자 내에 있으므로 다른 코드가 실행되기 전에 먼저 파일이 로드됩니다. XML 파일 로드가 완료되면 다음 코드와 같이 곧바로 `xmlCompleteHandler()` 메서드가 호출되어 외부 파일을 XML 객체로 파싱합니다.

```
private function xmlCompleteHandler(event:Event):void
{
  playlist = XML(event.target.data);
  videosXML = playlist.video;
  main();
}
```

재생 목록 XML 객체에는 외부 파일에서 가지온 원시 XML이 포함되어 있고, videosXML은 해당 비디오 노드가 포함된 XMLList 객체입니다. 다음 코드에서 샘플 playlist.xml 파일을 확인할 수 있습니다.

```
<videos>
  <video url="video/caption_video.flv" />
  <video url="video/cuepoints.flv" />
  <video url="video/water.flv" />
</videos>
```

마지막으로, `xmlCompleteHandler()` 메서드는 외부 FLV 파일 로드에 사용되는 NetConnection 객체 및 NetStream 객체뿐만 아니라 표시 목록의 다양한 구성 요소 인스턴스를 설정하는 `main()` 메서드도 호출합니다.

사용자 인터페이스 만들기

사용자 인터페이스를 만들려면 Button 인스턴스 다섯 개를 표시 목록으로 드래그하여 각 인스턴스에 `playButton`, `pauseButton`, `stopButton`, `backButton`, `forwardButton`이라는 이름을 지정해야 합니다.

다음 코드와 같이, 각 Button 인스턴스에 대해 `click` 이벤트에 대한 핸들러를 할당해야 합니다.

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

`buttonClickHandler()` 메서드는 다음 코드와 같이 switch 문을 사용하여 클릭된 Button 인스턴스를 확인합니다.

```
private function buttonClickHandler(event:MouseEvent):void
{
  switch (event.currentTarget)
  {
    case playButton:
      ns.resume();
      break;
    case pauseButton:
      ns.togglePause();
      break;
    case stopButton:
      ns.pause();
      ns.seek(0);
      break;
    case backButton:
      playPreviousVideo();
      break;
    case forwardButton:
      playNextVideo();
```

```
        break;
    }
}
```

그런 다음, Slider 인스턴스를 표시 목록에 추가하고 volumeSlider라는 인스턴스 이름을 지정합니다. 다음 코드는 Slider 인스턴스의 liveDragging 속성을 true로 설정하고 Slider 인스턴스의 change 이벤트에 대한 이벤트 리스너를 정의합니다.

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

ProgressBar 인스턴스를 표시 목록에 추가하고 positionBar라는 인스턴스 이름을 지정합니다. 다음 코드와 같이 mode 속성을 manual로 설정합니다.

```
positionBar.mode = ProgressBarMode.MANUAL;
```

마지막으로, Label 인스턴스를 표시 목록에 추가하고 positionLabel이라는 인스턴스 이름을 지정합니다. Timer 인스턴스에서 이 Label 인스턴스의 값을 설정합니다.

Video 객체의 메타데이터 수신

Flash Player에서 로드된 각 비디오에 대한 메타데이터가 발견되면 NetStream 객체의 client 속성에 대해 onMetaData() 콜백 핸들러가 호출됩니다. 다음 코드에서는 Object를 초기화하고 지정된 콜백 핸들러를 설정합니다.

```
client = new Object();
client.onMetaData = metadataHandler;
```

metadataHandler() 메서드는 데이터를 코드 내에 이전에 정의되어 있던 메타 속성에 복사합니다. 그러면 전체 응용 프로그램에서 언제든지 현재 비디오의 메타데이터에 액세스할 수 있습니다. 다음으로 스테이지에 있는 Video 객체의 크기가 메타데이터에서 반환된 크기에 맞게 조절됩니다. 마지막으로 현재 재생 중인 비디오의 크기에 맞게 positionBar 진행률 막대 인스턴스가 이동되고 크기가 조절됩니다. 다음 코드에는 전체 metadataHandler() 메서드가 포함되어 있습니다.

```
private function metadataHandler(metadataObj:Object):void
{
  meta = metadataObj;
  vid.width = meta.width;
  vid.height = meta.height;
  positionBar.move(vid.x, vid.y + vid.height);
  positionBar.width = vid.width;
}
```

동적 Flash 비디오 로드

각 Flash 비디오를 동적으로 로드하기 위해 응용 프로그램에서 NetConnection 및 NetStream 객체를 사용합니다. 다음 코드는 NetConnection 객체를 생성하고 `null` 값을 `connect()` 메서드에 전달합니다. Flash Player는 `null`을 지정하여 Flash Media Server 등의 서버에 연결하는 것이 아니라 로컬 서버에 있는 비디오에 연결합니다.

다음 코드는 NetConnection 인스턴스 및 NetStream 인스턴스를 모두 만들고, `netStatus` 이벤트에 대한 이벤트 리스너를 정의하며, `client` 객체를 `client` 속성에 할당합니다.

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

비디오의 상태가 변경될 때마다 `netStatusHandler()` 메서드가 호출됩니다. 비디오가 재생을 시작하거나 중지하는 경우, 비디오를 버퍼링하는 경우 또는 비디오 스트림을 찾을 수 없는 경우 등이 여기에 해당합니다. 다음 코드는 `netStatusHandler()` 이벤트를 나열합니다.

```
private function netStatusHandler(event:NetStatusEvent):void
{
  try
  {
    switch (event.info.code)
    {
      case "NetStream.Play.Start":
        t.start();
        break;
      case "NetStream.Play.StreamNotFound":
      case "NetStream.Play.Stop":
        t.stop();
        playNextVideo();
        break;
    }
  }
  catch (error:TypeError)
  {
    // 모든 오류를 무시합니다 .
  }
}
```

이전 코드는 Info 객체의 코드 속성을 평가하여 코드를 "NetStream.Play.Start", "NetStream.Play.StreamNotFound" 또는 "NetStream.Play.Stop"으로 필터링합니다. 다른 코드는 모두 무시됩니다. 넷스트림이 시작되면 코드에서 재생 헤드를 업데이트하는 Timer 인스턴스를 시작합니다. 넷스트림이 없거나 중지되면 Timer 인스턴스가 중지되고 응용 프로그램에서 재생 목록의 다음 비디오를 재생하려고 합니다.

Timer가 실행될 때마다 positionBar 진행률 막대 인스턴스는 ProgressBar 클래스의 setProgress() 메서드를 호출하여 현재 위치를 업데이트하고, positionLabel Label 인스턴스에는 현재 비디오의 경과 시간 및 총 시간이 업데이트됩니다.

```
private function timerHandler(event:TimerEvent):void
{
  try
  {
    positionBar.setProgress(ns.time, meta.duration);
    positionLabel.text = ns.time.toFixed(1) + " of "
  meta.duration.toFixed(1) + " seconds";
  }
  catch (error:Error)
  {
    // 이 오류를 무시합니다 .
  }
}
```

비디오 볼륨 제어

동적으로 로드된 비디오에 대해 NetStream 객체의 soundTransform 속성을 설정하여 볼륨을 제어할 수 있습니다. 비디오 주크박스 응용 프로그램을 활용하면 volumeSlider Slider 인스턴스 값을 변경하여 볼륨 수준을 수정할 수 있습니다. 다음 코드는 NetStream 객체의 soundTransform 속성으로 설정되어 있는 SoundTransform 객체에 Slider 구성 요소 값을 할당하여 볼륨 수준을 변경하는 방법을 보여 줍니다.

```
private function volumeChangeHandler(event:SliderEvent):void
{
  volumeTransform.volume = event.value;
  ns.soundTransform = volumeTransform;
}
```

비디오 재생 제어

응용 프로그램의 나머지 부분은 비디오가 비디오 스트림의 끝에 도달하거나 사용자가 이전 또는 다음 비디오를 건너뛸 때 비디오 재생을 제어합니다.

다음 메서드에서는 현재 선택된 인덱스의 XMLList에서 비디오 URL을 검색합니다.

```
private function getVideo():String
{
  return videosXML[idx].@url;
}
```

playVideo() 메서드는 NetStream 객체의 play() 메서드를 호출하여 현재 선택되어 있는 비디오를 로드합니다.

```
private function playVideo():void
{
```

```
  var url:String = getVideo();
  ns.play(url);
}
```

playPreviousVideo() 메서드는 현재 비디오 인덱스를 감소시키고, playVideo() 메서드를 호출하여 새 비디오 파일을 로드하며, 진행률 막대를 표시합니다.

```
private function playPreviousVideo():void
{
  if (idx > 0)
  {
    idx--;
    playVideo();
    positionBar.visible = true;
  }
}
```

마지막 메서드인 playNextVideo()는 비디오 인덱스를 증가시키고 playVideo() 메서드를 호출합니다. 현재 비디오가 재생 목록의 마지막 비디오인 경우, Video 객체에 대해 clear() 메서드가 호출되고 진행률 막대 인스턴스의 visible 속성이 false로 설정됩니다.

```
private function playNextVideo():void
{
  if (idx < (videosXML.length() - 1))
  {
    idx++;
    playVideo();
    positionBar.visible = true;
  }
  else
  {
    idx++;
    vid.clear();
    positionBar.visible = false;
  }
}
```

제 20 장

20 사운드를 사용한 작업

ActionScript로 대화식 몰입형 응용 프로그램을 만들 수 있습니다. 강력한 몰입형 응용 프로그램에서 자주 간과되는 요소가 사운드인데 비디오 게임에 사운드 효과를 추가하거나, 응용 프로그램 사용자 인터페이스에 오디오 피드백을 추가하거나, 인터넷에서 로드한 mp3 파일을 분석하는 프로그램을 만드는 등 사운드를 응용 프로그램에 중심적으로 활용할 수 있습니다.

이 장에서는 외부 오디오 파일의 로딩과 SWF에 포함된 오디오 관련 작업에 대해 설명합니다. 오디오 제어 방법, 사운드 정보의 시각적 표현 생성 방법, 마이크로부터 사운드를 캡처하는 방법 등이 소개됩니다.

목차

사운드를 사용한 작업의 기초

사운드를 사용한 작업 소개

컴퓨터에서 이미지를 디지털 포맷으로 인코딩하여 저장하고 다시 불러와 표시할 수 있는 것처럼, 컴퓨터에서 디지털 오디오를 캡처하고 인코딩(사운드 정보의 디지털화)하여 저장한 후 다시 불러와서 컴퓨터에 연결된 스피커를 통해 재생할 수 있습니다. 사운드 재생 방법 중 하나는 Adobe Flash Player 및 ActionScript를 사용하는 것입니다.

사운드 데이터가 디지털 형식으로 변환되면 사운드 볼륨과 스테레오 또는 모노 사운드 등의 다양한 특성이 생깁니다. ActionScript에서 사운드를 재생할 때 이러한 특성도 조정할 수 있습니다. 예를 들어, 사운드를 더 크게 하거나 특정 방향에서 사운드가 나오는 듯한 효과를 연출할 수 있습니다.

ActionScript에서 사운드를 제어하려면 먼저 사운드 정보를 Flash Player에 로드해야 합니다. ActionScript를 사용하여 작업할 수 있도록 오디오 데이터를 Flash Player에 로드하는 방법은 4가지가 있습니다. 첫째, 외부 사운드 파일(예: mp3 파일)을 SWF에 로드하거나 둘째, SWF 파일을 만들 때 사운드 정보를 직접 SWF 파일에 포함하거나 셋째, 컴퓨터에 연결된 마이크를 사용하여 오디오를 입력하거나 넷째, 서버에서 스트리밍한 사운드 데이터에 액세스할 수 있습니다.

외부 사운드 파일로부터 사운드 데이터를 로드할 때는 나머지 사운드 데이터를 로드하는 중에도 사운드 파일의 시작 부분을 먼저 재생할 수 있습니다.

디지털 오디오를 인코딩하는 데 사용되는 사운드 파일 포맷은 다양하지만 ActionScript 3.0 및 Flash Player는 mp3 포맷으로 저장된 사운드 파일을 지원합니다. WAV 또는 AIFF 등의 다른 포맷으로 된 사운드 파일은 직접 로드하거나 재생할 수 없습니다.

ActionScript에서 사운드 관련 작업을 하는 동안에는 flash.media 패키지의 몇 가지 클래스를 다루게 될 것입니다. Sound 클래스는 사운드 파일을 로드하고 재생을 시작하여 오디오 정보에 대한 액세스 권한을 얻는 데 사용하는 클래스입니다. 사운드 재생을 시작하면 Flash Player 사용자에게 SoundChannel 객체에 대한 액세스 권한이 부여됩니다. 로드한 오디오 파일은 사용자의 컴퓨터에서 재생하는 몇 가지 사운드 중 하나일 수 있으므로 재생되는 각 개별 사운드는 자체의 SoundChannel 객체를 사용합니다. 함께 믹싱되는 모든 SoundChannel 객체의 조합된 출력은 컴퓨터 스피커를 통해 실제로 재생되는 사운드입니다. 이 SoundChannel 인스턴스를 사용하여 사운드의 속성을 제어하거나 재생을 중지할 수 있습니다. 마지막으로, 조합된 오디오를 제어하려는 경우 SoundMixer 클래스를 통해 믹싱된 출력을 제어할 수 있습니다.

ActionScript에서 사운드 관련 작업을 할 때 기타 몇 가지 클래스를 사용하여 특정 작업을 수행할 수도 있습니다. 사운드 관련 클래스 전반에 대한 자세한 내용은 558페이지의 “사운드 아키텍처의 이해”를 참조하십시오.

사운드와 관련된 일반적인 작업

이 장에서는 사용자가 수행할 다음과 같은 사운드 관련 작업에 대해 설명합니다.

- 외부 mp3 파일 로드 및 로드 진행률 추적
- 사운드의 재생, 일시 정지, 다시 시작 및 중지
- 로드하는 동안 스트리밍 사운드 재생
- 사운드 볼륨 및 패닝 조작
- mp3 파일에서 ID3 메타데이터 검색
- 원시 사운드 웨이브 데이터 사용
- 사용자의 마이크에서 사운드 입력 캡처 및 재생

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 진폭: 사운드 파형의 한 점과 0 또는 기준선 사이의 거리입니다.
- 비트율: 사운드 파일의 초당 인코딩 또는 스트리밍된 데이터의 양입니다. mp3 파일의 경우 비트율은 보통 kbps(초당 킬로비트) 단위로 표현됩니다. 비트율이 높을수록 일반적으로 음질이 좋습니다.
- 버퍼링: 사운드 재생 전에 사운드 데이터를 수신하여 저장하는 과정입니다.
- mp3: MPEG-1 Audio Layer 3, 즉 mp3는 널리 사용되는 사운드 압축 포맷입니다.
- 패닝: 스테레오 사운드 필드의 왼쪽과 오른쪽 채널 간에 오디오 신호를 지정하는 과정입니다.
- 피크: 파형에서 가장 높은 위치입니다.
- 샘플링 속도: 디지털 신호를 만들기 위해 아날로그 오디오 신호로부터 추출한 초당 샘플링 수를 정의합니다. 표준 콤팩트 디스크 오디오의 샘플링 속도는 44.1kHz, 즉 44,100샘플링/초입니다.
- 스트리밍: 사운드 파일 또는 비디오 파일의 뒷 부분을 서버에서 로드하는 동안, 이미 로드된 앞 부분을 재생하는 프로세스입니다.
- 볼륨: 사운드의 크기입니다.
- 파형: 시간에 따른 사운드 신호의 진폭 변화를 나타내는 그래프 모양입니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드 목록을 직접 테스트할 수 있습니다. 이 장은 ActionScript에서의 사운드 관련 작업을 다루므로, 대부분의 예제가 사운드 파일 작업(파일 재생, 재생 중지, 사운드 조절 등)과 관련되어 있습니다. 이 장의 예제를 테스트하려면:

1. 새 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 타임라인에서 첫 번째 키프레임을 선택하고 [액션] 패널을 엽니다.
3. [스크립트] 창에 예제 코드 샘플을 복사합니다.
4. 코드에서 외부 사운드 파일을 로드할 경우 다음과 같은 코드 행이 포함됩니다.

   ```
   var req:URLRequest = new URLRequest("click.mp3");
   var s:Sound = new Sound(req);
   ```

   여기서 "click.mp3"는 로드될 사운드 파일의 이름입니다. 이러한 예제를 테스트하려면 사용할 mp3 파일이 있어야 하며, 이 mp3 파일을 Flash 문서와 동일한 폴더에 넣어야 합니다. 그런 다음, 코드의 이름을 코드 샘플에 있는 이름 대신 mp3 파일 이름으로 바꿉니다. 예를 들어, 위의 코드의 경우 "click.mp3"라는 이름을 해당 mp3 파일의 이름으로 변경합니다.
5. 주 메뉴에서 [컨트롤] > [무비 테스트]를 선택하여 SWF 파일을 만들고, 예제의 결과를 미리 보고 듣습니다.

일부 예제는 오디오 재생뿐만 아니라 `trace()` 함수를 사용하여 값을 나타내기도 합니다. 이러한 예제를 테스트할 경우 해당 값의 결과를 [출력] 패널에서 확인할 수 있습니다. 일부 예제는 내용을 스크린에 그리므로, 이러한 예제의 경우에는 해당 내용을 Flash Player 윈도우에서도 볼 수 있습니다.

이 설명서의 예제 코드 샘플 테스트에 대한 자세한 내용은 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"를 참조하십시오.

사운드 아키텍처의 이해

응용 프로그램은 다음과 같은 4가지 소스로부터 사운드 데이터를 로드할 수 있습니다.

- 런타임 시 로드된 외부 사운드 파일
- 응용 프로그램의 SWF 파일에 포함된 사운드 리소스
- 사용자의 시스템에 연결된 마이크로부터 입력된 사운드 데이터
- Flash Media Server 등의 원격 미디어 서버로부터 스트리밍된 사운드 데이터

사운드 데이터는 완전히 로드된 후에 재생되거나 스트리밍(로드 중 재생)될 수 있습니다.

ActionScript 3.0 및 Flash Player는 mp3 포맷으로 저장된 사운드 파일을 지원합니다. WAV 또는 AIFF 등의 다른 포맷으로 된 사운드 파일은 직접 로드하거나 재생할 수 없습니다.

Adobe Flash CS3 Professional을 사용하면 WAV 또는 AIFF 사운드 파일을 가져와서 이 파일을 mp3 포맷으로 사용자 응용 프로그램의 SWF 파일에 포함할 수 있습니다. Flash 제작 도구를 사용하면 포함된 사운드 파일을 압축하여 파일 크기를 줄일 수도 있습니다. 단, 파일 크기가 작아지면 사운드 품질이 저하됩니다. 자세한 내용은 *Flash 사용 설명서*의 "사운드 가져오기"를 참조하십시오.

ActionScript 3.0 사운드 아키텍처는 flash.media 패키지에 있는 다음 클래스를 사용합니다.

클래스	설명
flash.media.Sound	Sound 클래스는 사운드의 로드를 처리하고 기본적인 사운드 속성을 관리하며 사운드 재생을 시작합니다.
flash.media.SoundChannel	응용 프로그램이 Sound 객체를 재생할 때는 새로운 SoundChannel 객체가 만들어져 재생을 제어합니다. SoundChannel 객체는 사운드의 왼쪽과 오른쪽 재생 채널의 볼륨을 제어합니다. 재생되는 각 사운드에는 자체의 SoundChannel 객체가 있습니다.
flash.media.SoundLoaderContext	SoundLoaderContext 클래스는 사운드를 로드할 때 사용할 버퍼링 시간(초)을 지정하고 파일을 로드할 때 서버에서 크로스 도메인 정책 파일을 찾을지 여부를 지정합니다. SoundLoaderContext 객체는 `Sound.load()` 메서드에 대한 매개 변수로 사용됩니다.
flash.media.SoundMixer	SoundMixer 클래스는 응용 프로그램의 모든 사운드와 관련된 보안 속성 및 재생을 제어합니다. 사실상, 다중 사운드 채널은 공통 SoundMixer 객체를 통해 믹싱되므로 SoundMixer 객체의 속성 값은 현재 재생되는 모든 SoundChannel 객체에 영향을 미칩니다.
flash.media.SoundTransform	SoundTransform 클래스에는 사운드 볼륨과 패닝을 제어하는 값이 포함됩니다. SoundTransform 객체는 무엇보다 개별 SoundChannel 객체, 전역 SoundMixer 객체 또는 Microphone 객체에 적용될 수 있습니다.
flash.media.ID3Info	ID3Info 객체에는 종종 mp3 사운드 파일에 저장되는 ID3 메타데이터 정보를 나타내는 속성이 포함됩니다.
flash.media.Microphone	Microphone 클래스는 사용자의 컴퓨터에 연결된 마이크 또는 기타 사운드 입력 장치를 나타냅니다. 마이크로부터 입력된 오디오는 로컬 스피커나 원격 서버로 라우팅될 수 있습니다. Microphone 객체는 게인, 샘플링 속도 및 자체 사운드 스트림의 기타 특성을 제어합니다.

로드 및 재생되는 각 사운드에는 Sound 클래스 및 SoundChannel 클래스의 고유한 인스턴스가 필요합니다. 그리고 다중 SoundChannel 인스턴스의 출력이 재생 중에 전역 SoundMixer 클래스에 의해 함께 믹싱됩니다.

Sound, SoundChannel, SoundMixer 클래스는 마이크를 통해 캡처한 사운드 데이터 또는 Flash Media Server 등의 스트리밍 미디어에서 캡처한 사운드 데이터에 사용되지 않습니다.

외부 사운드 파일 로드

Sound 클래스의 각 인스턴스는 특정 사운드 리소스의 재생을 로드 및 트리거하기 위한 것입니다. 응용 프로그램에서 Sound 객체를 다시 사용하여 둘 이상의 사운드를 로드할 수 없습니다. 새로운 사운드 리소스를 로드하려면 새로운 Sound 객체를 만들어야 합니다.

작은 사운드 파일(예: 버튼에 연결될 클릭 사운드)을 로드하는 경우 응용 프로그램에서 새로운 Sound를 만들어 아래와 같이 사운드 파일을 자동으로 로드할 수 있습니다.

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

`Sound()` 생성자는 URLRequest 객체를 첫 번째 매개 변수로 받습니다. URLRequest 매개 변수의 값이 제공되면 새로운 Sound 객체는 지정된 사운드 리소스를 자동으로 로드합니다.

아주 단순한 경우를 제외하고는 응용 프로그램은 사운드의 로드 진행률을 살피고 로드 중에 오류가 발생하지 않는지 확인해야 합니다. 예를 들어, 클릭 사운드의 용량이 꽤 큰 경우 사운드를 트리거하는 버튼을 사용자가 클릭할 때 사운드가 완전히 로드되지 않을 수 있습니다. 로드되지 않은 사운드를 재생하려고 하면 런타임 오류가 발생할 수 있습니다. 사운드 재생을 시작하는 액션을 취하도록 허용하기 전에 사운드가 완전히 로드될 때까지 기다리는 것이 안전합니다.

Sound 객체는 사운드 로드 프로세스 중에 여러 가지 이벤트를 전달합니다. 응용 프로그램은 이러한 이벤트를 수신하여 로드 진행률을 추적하고 사운드 재생 전에 사운드가 완전히 로드되도록 할 수 있습니다. 다음 표에서는 Sound 객체가 전달할 수 있는 이벤트를 설명합니다.

이벤트	설명
open(`Event.OPEN`)	사운드 로드 작업이 시작되기 직전에 전달됩니다.
progress(`ProgressEvent.PROGRESS`)	사운드 로드 프로세스 중에 파일 또는 스트림으로부터 데이터를 수신할 때 정기적으로 전달됩니다.
id3(`Event.ID3`)	mp3 사운드에 대해 ID3 데이터를 사용할 수 있는 경우 전달됩니다.
complete(`Event.COMPLETE`)	모든 사운드 리소스 데이터가 로드되었을 때 전달됩니다.
ioError(`IOErrorEvent.IO_ERROR`)	사운드 파일을 찾을 수 없거나 모든 사운드 데이터를 수신하기 전에 로드 프로세스가 중단된 경우 전달됩니다.

다음 코드는 사운드 로드 완료 후 재생 방법을 보여 줍니다.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;
```

```
var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

이 코드 샘플은 먼저 새 Sound 객체를 만듭니다. 이때 이 객체에 URLRequest 매개 변수에 대한 초기 값을 지정하지 않습니다. 그런 다음 Sound 객체로부터 `Event.COMPLETE` 이벤트를 수신하고, 모든 사운드 데이터가 로드되면 `onSoundLoaded()` 메서드를 실행합니다. 다음으로, 사운드 파일에 대한 새로운 URLRequest 값을 사용하여 `Sound.load()` 메서드를 호출합니다.

사운드 로드가 완료되면 `onSoundLoaded()` 메서드가 실행됩니다. Event 객체의 대상 속성은 Sound 객체에 대한 참조입니다. Sound 객체의 `play()` 메서드를 호출하면 사운드 재생이 시작됩니다.

사운드 로드 프로세스 모니터링

사운드 파일은 용량이 너무 커서 로드하는 데 오랜 시간이 걸릴 수 있습니다. Flash Player를 사용하면 사운드를 완전히 로드하기 전에도 응용 프로그램에서 사운드를 재생할 수 있으며 로드된 사운드 데이터의 양과 이미 재생된 사운드의 양을 표시할 수도 있습니다.

Sound 클래스는 사운드의 로드 진행률을 상대적으로 쉽게 표시하도록 하는 두 개의 이벤트인 `ProgressEvent.PROGRESS` 및 `Event.COMPLETE`를 전달합니다. 다음 예제는 이러한 이벤트를 사용하여 로드되는 사운드에 대한 진행률 정보를 표시하는 방법을 보여 줍니다.

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
  var loadedPct:uint =
    Math.round(100 * (event.bytesLoaded / event.bytesTotal));
```

```
  trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
  var localSound:Sound = event.target as Sound;
  localSound.play();
}
function onIOError(event:IOErrorEvent)
{
  trace("The sound could not be loaded: " + event.text);
}
```

이 코드는 Sound 객체를 만든 다음 ProgressEvent.PROGRESS 및 Event.COMPLETE 이벤트에 대해 리스너를 이 객체에 추가합니다. Sound.load() 메서드가 호출되고 첫 번째 데이터가 사운드 파일로부터 수신된 후에 ProgressEvent.PROGRESS 이벤트가 발생하고 onSoundLoadProgress() 메서드가 트리거됩니다.

로드된 사운드 데이터의 백분율은 ProgressEvent 객체의 bytesLoaded 속성 값을 bytesTotal 속성 값으로 나눈 값과 같습니다. 동일한 bytesLoaded 및 bytesTotal 속성은 Sound 객체에서도 사용 가능합니다. 위 예제에서는 사운드 로드 진행률에 대한 메시지만 보여 주지만, 사용자가 bytesLoaded 및 bytesTotal 값을 사용하여 진행률 막대 구성 요소(Adobe Flex 2 프레임워크 또는 Flash 제작 도구와 함께 제공됨)를 쉽게 업데이트할 수 있습니다.

또한 이 예제는 사운드 파일을 로드할 때 응용 프로그램이 오류를 어떻게 인식하고 이에 응답하는지 보여 줍니다. 예를 들어 특정 파일 이름을 가진 사운드 파일을 찾을 수 없는 경우, Sound 객체가 Event.IO_ERROR 이벤트를 전달합니다. 이전 코드에서는 오류가 발생할 때 onIOError() 메서드가 실행되고 간단한 오류 메시지가 표시됩니다.

포함된 사운드를 사용한 작업

외부 파일로부터 사운드를 로드하는 대신 포함된 사운드를 사용하는 것은 응용 프로그램의 사용자 인터페이스에서 지시자로 사용되는 작은 사운드(예: 버튼을 클릭했을 때 재생되는 사운드)에 가장 유용합니다.

사운드 파일을 응용 프로그램에 포함시키면 SWF 파일의 크기는 사운드 파일의 크기만큼 증가합니다. 즉, 응용 프로그램에 대용량 사운드 파일을 포함시키면 SWF 파일의 크기가 지나치게 커질 수 있습니다.

사운드 파일을 응용 프로그램의 SWF 파일에 포함시키는 정확한 방법은 개발 환경에 따라 다릅니다.

포함된 사운드 파일을 Flash에서 사용

Flash 제작 도구를 사용하면 다양한 사운드 포맷의 사운드를 가져와서 라이브러리에 심볼로 저장할 수 있습니다. 그런 다음, 사운드를 타임라인 내의 프레임 또는 버튼 상태의 프레임에 할당한 후 비헤이비어와 함께 사용하거나, 사운드를 직접 ActionScript 코드에서 사용할 수도 있습니다. 이 단원에서는 Flash 제작 도구를 활용하여 ActionScript 코드에 포함된 사운드를 사용하는 방법에 대해 설명합니다. 포함된 사운드를 Flash에서 사용하는 기타 방법에 대한 자세한 내용은 *Flash 사용 설명서*의 "사운드 가져오기"를 참조하십시오.

사운드 파일을 Flash 무비에 포함하려면:

1. [파일] > [가져오기] > [라이브러리로 가져오기]를 선택한 다음 사운드 파일을 하나 선택하여 가져옵니다.
2. 가져온 파일 이름을 [라이브러리] 패널에서 마우스 오른쪽 버튼으로 클릭한 후 [속성]을 선택합니다. [ActionScript에 내보내기] 체크 상자를 클릭합니다.
3. [클래스] 필드에 이 포함된 사운드를 ActionScript에서 참조할 때 사용할 이름을 입력합니다. 기본적으로 이 필드의 사운드 파일 이름이 사용됩니다. "DrumSound.mp3"의 경우와 같이 파일 이름에 마침표가 포함된 경우 "DrumSound" 등으로 변경해야 합니다. ActionScript는 클래스 이름에 마침표 문자를 허용하지 않습니다. [기본 클래스] 필드에는 flash.media.Sound가 여전히 표시됩니다.
4. [확인]을 클릭합니다. 이 클래스에 대한 정의를 클래스 경로에서 찾을 수 없다는 대화 상자가 나타날 수 있습니다. [확인]을 클릭하고 계속합니다. 사용자 응용 프로그램의 클래스 경로에 있는 클래스 이름과 일치하지 않는 클래스 이름을 입력하는 경우, flash.media.Sound 클래스에서 상속된 새 클래스가 자동으로 생성됩니다.
5. 포함된 사운드를 사용하려면 이 사운드에 대한 클래스 이름을 ActionScript에서 참조합니다. 예를 들어, 다음 코드는 자동으로 생성된 DrumSound 클래스의 새 인스턴스를 만듭니다.

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound는 flash.media.Sound 클래스의 하위 클래스이므로 위에 나와 있는 `play()` 메서드를 포함하여 Sound 클래스의 메서드와 속성을 상속합니다.

사운드 파일 스트리밍 작업

사운드 파일이나 비디오 파일의 데이터를 로드하고 있는 중에 먼저 로드된 부분을 재생하는 것을 *스트리밍*이라고 합니다. 원격 서버로부터 로드되는 외부 사운드 파일은 스트리밍되는 경우가 많으므로 전체 사운드 데이터의 로드가 완료되지 않아도 사운드를 들을 수 있습니다.

`SoundMixer.bufferTime` 속성은 사운드 재생 전에 Flash Player가 수집해야 하는 사운드 데이터의 밀리초를 나타냅니다. 즉, `bufferTime` 속성이 5000인 경우 Flash Player가 사운드 파일로부터 최소한 5000밀리초 상당의 데이터를 로드해야 사운드 재생을 시작할 수 있습니다. 기본 `SoundMixer.bufferTime` 값은 1000입니다.

응용 프로그램은 사운드를 로드할 때 새로운 `bufferTime` 값을 명시적으로 지정하여 개별 사운드에 대한 전역 `SoundMixer.bufferTime` 값을 재정의할 수 있습니다. 기본 버퍼링 시간을 재정의하려면 아래와 같이 먼저 SoundLoaderContext 클래스의 새 인스턴스를 만들고 해당 `bufferTime` 속성을 설정하여 `Sound.load()` 메서드에 매개 변수로 전달합니다.

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

재생을 계속하는 동안 Flash Player는 사운드 버퍼 크기를 동일 수준 이상으로 유지하려고 합니다. 사운드 데이터가 재생 속도보다 빠르게 로드되면 재생은 중단 없이 계속됩니다. 하지만 네트워크 제한으로 인해 데이터 로드 속도가 저하되는 경우 재생 헤드가 사운드 버퍼의 끝에 도달할 수 있습니다. 이렇게 되면 재생이 일시 중단되지만 추가 사운드 데이터가 로드되면 재생이 자동으로 다시 시작됩니다.

Flash Player가 데이터 로드를 대기하느라 재생이 일시 중단되었는지 확인하려면 `Sound.isBuffering` 속성을 사용합니다.

사운드 재생

로드된 사운드의 재생은 아래와 같이 Sound 객체에 대한 `Sound.play()` 메서드를 호출하는 것처럼 간단할 수 있습니다.

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

ActionScript 3.0을 사용하여 사운드를 재생할 경우 다음 작업을 수행할 수 있습니다.

- 특정 시작 위치에서 사운드 재생
- 사운드를 일시 정지하고 나중에 같은 위치에서 다시 재생 시작

- 사운드 재생이 끝나는 정확한 시간 알아보기
- 사운드의 재생 진행률 추적
- 사운드 재생 중에 볼륨 변경 및 패닝

재생 중에 이러한 작업을 수행하려면 SoundChannel, SoundMixer 및 SoundTransform 클래스를 사용합니다.

SoundChannel 클래스는 단일 사운드의 재생을 제어합니다. `SoundChannel.position` 속성은 재생 헤드로 간주할 수 있으며 재생 중인 사운드 데이터의 현재 위치를 나타냅니다.

응용 프로그램이 `Sound.play()` 메서드를 호출하면 SoundChannel 클래스의 새로운 인스턴스가 만들어져 재생을 제어합니다.

응용 프로그램은 특정 시작 위치로부터 해당 위치를 지나면서 사운드를 재생할 수 있습니다. 시작 위치는 기준이 밀리초이며 `Sound.play()` 메서드의 `startTime` 매개 변수로 지정됩니다. `Sound.play()` 메서드의 loops 매개 변수에 숫자 값을 전달하여 사운드를 연속으로 빠르게 반복할 고정 횟수를 지정할 수도 있습니다.

`Sound.play()` 메서드가 `startTime` 매개 변수 및 `loops` 매개 변수를 사용하여 호출되면 다음 코드에서 볼 수 있는 것처럼 사운드가 매번 동일한 시작 위치에서 반복적으로 재생됩니다.

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

위 예제에서는 사운드가 시작된지 1초 후 시점에 재생되며 연속 세 번 반복됩니다.

사운드 정지 및 다시 시작

응용 프로그램이 노래나 포드캐스트 등의 긴 사운드를 재생하는 경우 이러한 사운드의 재생을 일시 정지 및 다시 시작하도록 할 수 있습니다. ActionScript에서는 엄밀한 의미로 사운드를 재생 중에 일시 정지할 수는 없고 중단만 가능합니다. 하지만 어느 위치에서든 사운드 재생을 시작할 수 있습니다. 사운드가 중지된 시점의 위치를 기록하여 나중에 해당 위치에서 사운드 재생을 시작할 수 있습니다.

예를 들어, 코드가 다음과 같이 사운드 파일을 로드하여 재생한다고 가정합니다.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

사운드가 재생되는 동안 `SoundChannel.position` 속성은 사운드 파일에서 현재 재생 중인 위치를 나타냅니다. 응용 프로그램은 다음과 같이 사운드 재생이 중지되기 전에 위치 값을 저장할 수 있습니다.

```
var pausePosition:int = channel.position;
channel.stop();
```

사운드 재생을 다시 시작하려면 이전에 저장된 위치 값을 전달하여 사운드가 중지되었던 위치에서 다시 시작합니다.

```
channel = snd.play(pausePosition);
```

재생 모니터링

응용 프로그램은 사운드 재생이 언제 중지되는지 인식하여 다른 사운드 재생을 시작하거나 이전 재생 중에 사용된 일부 리소스를 정리할 수 있습니다. SoundChannel 클래스는 사운드 재생이 끝날 때 `Event.SOUND_COMPLETE` 이벤트를 전달합니다. 응용 프로그램은 아래와 같이 이 이벤트를 수신하여 적절한 액션을 취할 수 있습니다.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound("smallSound.mp3");
var channel:SoundChannel = snd.play();
s.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
}
```

SoundChannel 클래스는 재생 중에 진행률 이벤트를 전달하지 않습니다. 재생 진행률을 보고하려면 응용 프로그램은 자체의 시간 메커니즘을 설정하고 사운드 재생 헤드의 위치를 추적해야 합니다.

재생된 사운드 백분율을 계산하려면 다음과 같이 `SoundChannel.position` 속성 값을 재생 중인 사운드 데이터의 길이로 나누어야 합니다.

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

이 코드는 재생 시작 전에 사운드 데이터가 완전히 로드된 경우에만 정확한 재생 백분율을 보고합니다. `Sound.length` 속성은 전체 사운드 파일의 최종 크기가 아닌 현재 로드된 사운드 데이터의 크기를 표시합니다. 아직 로드 중인 스트리밍 사운드의 재생 진행률을 추적하기 위해 응용 프로그램은 전체 사운드 파일의 최종 크기를 추정하여 그 값을 계산에 사용해야 합니다. 다음과 같이 Sound 객체의 `bytesLoaded` 및 `bytesTotal` 속성을 사용하여 사운드 데이터의 최종 길이를 추정할 수 있습니다.

```
var estimatedLength:int =
  Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

다음 코드는 좀 더 큰 사운드 파일을 로드하며, 재생 진행률을 표시하는 데 `Event.ENTER_FRAME` 이벤트를 시간 메커니즘으로 사용합니다. 그리고 현재 위치 값을 사운드 데이터의 전체 길이로 나누어 계산되는 재생 백분율을 정기적으로 보고합니다.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
  URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
  var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
  var playbackPercent:uint =
    Math.round(100 * (channel.position / estimatedLength));
  trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

사운드 데이터 로드가 시작되고 나면 이 코드는 `snd.play()` 메서드를 호출하고 결과 SoundChannel 객체를 `channel` 변수에 저장합니다. 그런 다음 이벤트 리스너를 `Event.ENTER_FRAME` 이벤트에 대한 기본 응용 프로그램에 추가하고 또 다른 이벤트 리스너를 재생 완료 시 발생하는 `Event.SOUND_COMPLETE` 이벤트에 대한 SoundChannel 객체에 추가합니다.

응용 프로그램이 해당 애니메이션의 새로운 프레임에 도달할 때마다 `onEnterFrame()` 메서드가 호출됩니다. `onEnterFrame()` 메서드는 이미 로드된 데이터 양을 기준으로 사운드 파일의 전체 길이를 추정한 다음 현재 재생 백분율을 계산하여 표시합니다.

전체 사운드가 재생되었으면 `onPlaybackComplete()` 메서드가 실행되어 `Event.ENTER_FRAME` 이벤트에 대한 이벤트 리스너를 제거합니다. 그러면 재생이 완료된 후에 진행률 업데이트를 더 이상 표시하지 않습니다.

`Event.ENTER_FRAME` 이벤트는 초당 여러 번 전달될 수 있습니다. 어떤 경우에는 그렇게 빈번하게 재생 진행률을 표시할 필요가 없을 수 있습니다. 이런 경우에는 사용자의 응용 프로그램에서 flash.util.Timer 클래스를 사용하여 고유한 시간 메커니즘을 설정할 수 있습니다. 181페이지의 “날짜 및 시간을 사용한 작업”을 참조하십시오.

사운드 스트리밍 중단

스트리밍되는 사운드 즉, 재생하는 동안 계속 로드되는 사운드의 경우 재생 프로세스에 이상 현상이 발생하기도 합니다. 스트리밍 사운드를 재생 중인 SoundChannel 인스턴스에서 응용 프로그램이 `SoundChannel.stop()` 메서드를 호출하면 한 프레임에 대한 사운드 재생이 중지되고 다음 프레임에서 사운드를 처음부터 다시 재생합니다. 이러한 현상이 나타나는 이유는 사운드 로드 프로세스가 아직 진행 중이기 때문입니다. 스트리밍 사운드의 로드와 재생을 모두 중단하려면 `Sound.close()` 메서드를 호출합니다.

사운드 로드 및 재생 시의 보안 고려 사항

응용 프로그램의 사운드 데이터 액세스 기능은 Flash Player 보안 모델에 따라 제한됩니다. 각 사운드에는 두 가지 보안 샌드박스의 제한 사항이 적용됩니다. 보안 샌드박스는 내용 자체에 대한 샌드박스("내용 샌드박스")와 사운드를 로드하여 재생하는 응용 프로그램 또는 객체에 대한 샌드박스("소유자 샌드박스")가 있습니다. 전반적인 Flash Player 보안 모델 및 샌드박스 정의에 대한 자세한 내용은 707페이지의 "Flash Player 보안"을 참조하십시오.

내용 샌드박스는 `id3` 속성 또는 `SoundMixer.computeSpectrum()` 메서드를 사용하여 사운드에서 세부 사운드 데이터를 추출할 수 있는지 여부를 제어하는 것으로, 사운드 파일 자체의 로드 또는 재생을 제한하지는 않습니다.

사운드 파일의 원본 도메인은 내용 샌드박스의 보안 제한 사항을 정의합니다. 일반적으로 사운드 파일을 로드하는 응용 프로그램 또는 객체의 SWF 파일과 동일한 도메인 또는 폴더에 사운드 파일이 있으면 응용 프로그램 또는 객체는 해당 사운드 파일에 액세스할 수 있습니다. 응용 프로그램과는 다른 도메인에서 사운드가 제공되는 경우에도 크로스 도메인 정책 파일을 사용하여 내용 샌드박스에서 사운드를 가져올 수 있습니다.

응용 프로그램은 `checkPolicyFile` 속성을 사용하여 SoundLoaderContext 객체를 `Sound.load()` 메서드에 매개 변수로 전달할 수 있습니다. `checkPolicyFile` 속성을 `true`로 설정하면 Flash Player는 사운드가 로드된 서버에서 크로스 도메인 정책 파일을 찾습니다. 크로스 도메인 정책 파일이 존재하고 로드하는 SWF 파일의 도메인에 액세스 권한이 부여되면 SWF 파일은 사운드 파일을 로드하고 Sound 객체의 `id3` 속성에 액세스한 다음 로드된 사운드에 대한 `SoundMixer.computeSpectrum()` 메서드를 호출합니다.

소유자 샌드박스는 사운드의 로컬 재생을 제어합니다. 사운드 재생을 시작하는 응용 프로그램 또는 객체가 소유자 샌드박스를 정의합니다.

다음 조건에 맞는 경우 `SoundMixer.stopAll()` 메서드는 현재 재생 중인 모든 SoundChannel 객체의 사운드를 중지합니다.

- 동일한 소유자 샌드박스 내의 객체에 의해 시작된 사운드
- `SoundMixer.stopAll()` 메서드를 호출하는 응용 프로그램 또는 객체의 도메인에 대한 액세스를 허용하는 크로스 도메인 정책 파일을 가진 소스에서 제공되는 사운드

SoundMixer.stopAll() 메서드가 모든 재생 사운드를 실제로 중지하는지 알아보기 위해 응용 프로그램이 SoundMixer.areSoundsInaccessible() 메서드를 호출할 수 있습니다. 해당 메서드가 true 값을 반환하면 재생 중인 일부 사운드는 현재 소유자 샌드박스의 제어 범위를 벗어나므로 SoundMixer.stopAll() 메서드에 의해 중지되지 않습니다.

또한 SoundMixer.stopAll() 메서드는 외부 파일로부터 로드된 모든 사운드에 대해 재생 헤드가 계속 진행되지 않도록 중지합니다. 하지만 Flash 제작 도구를 사용하여 FLA 파일에 포함된 사운드 및 타임라인 내 프레임에 연결된 사운드는애니메이션이 새 프레임으로 이동하면 다시 재생되기 시작합니다.

사운드 볼륨 및 패닝 제어

개별 SoundChannel 객체는 사운드에 대해 왼쪽 및 오른쪽 스테레오 채널을 모두 제어합니다. mp3 사운드가 모노인 경우, SoundChannel 객체의 왼쪽 및 오른쪽 스테레오 채널에 동일한 파형이 포함됩니다.

SoundChannel 객체의 leftPeak 및 rightPeak 속성을 사용하여 재생 중인 사운드에 대한 각 스테레오 채널의 진폭을 알 수 있습니다. 이러한 속성은 사운드 파형 자체의 피크 진폭을 표시하는 것으로, 실제 재생 볼륨을 나타내지는 않습니다. 실제 재생 볼륨은 SoundChannel 객체 및 SoundMixer 클래스에 설정된 볼륨 값 및 사운드 웨이브 진폭의 함수입니다.

SoundChannel 객체의 팬 속성은 재생 중에 왼쪽 및 오른쪽 채널 각각에 다른 볼륨 수준을 지정하는 데 사용될 수 있습니다. 팬 속성은 -1 ~ 1 범위의 값을 가질 수 있습니다. 여기서 -1은 오른쪽 채널 음을 소거한 상태에서 왼쪽 채널을 최고 볼륨으로 재생한다는 의미이며, 1은 왼쪽 채널 음을 소거한 상태에서 오른쪽 채널을 최고 볼륨으로 재생한다는 의미입니다. -1과 1 사이의 숫자 값은 왼쪽과 오른쪽 채널 값에 대한 비례 값을 설정하며, 0은 양쪽 채널이 균형 있는 중간 볼륨 수준으로 재생된다는 의미입니다.

다음 코드 예제는 볼륨 값이 0.6이고 팬 값이 -1(즉, 왼쪽 채널이 최고 볼륨으로 재생되고 오른쪽 채널은 음소거)인 SoundTransform 객체를 만듭니다. 이 코드는 SoundTransform 객체를 play() 메서드에 매개 변수로 전달하여 재생 제어를 위해 만들어진 새로운 SoundChannel 객체에 해당 SoundTransform 객체를 적용합니다.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

사운드 재생 도중 SoundTransform 객체의 pan 속성이나 volume 속성을 설정한 다음, 이 객체를 SoundChannel 객체의 soundTransform 속성으로 적용하여 패닝 및 볼륨을 변경할 수 있습니다.

다음 예제에서처럼 SoundMixer 클래스의 soundTransform 속성을 사용하여 한번에 모든 사운드에 대한 전역 볼륨 값과 팬 값을 설정할 수도 있습니다.

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

또는 SoundTransform 객체를 사용하여 Microphone 객체에 대한(575페이지의 “사운드 입력 캡처” 참조) 볼륨 및 팬 값을 설정할 수도 있고, Sprite 객체와 SimpleButton 객체에 대해서도 설정할 수 있습니다.

다음 예제에서는 사운드 재생 중에 사운드의 패닝을 왼쪽 채널에서 오른쪽 채널로 교대하고 또 그 반대로 전환합니다.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
  trans.pan = Math.sin(panCounter);
  channel.soundTransform = trans; // 또는 SoundMixer.soundTransform = trans;
  panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

이 코드는 사운드 파일을 로드한 다음 볼륨이 1(최대 볼륨)로, 팬이 0(좌우 밸런스 동일)으로 설정된 새 SoundTransform 객체를 만듭니다. 그런 다음 `snd.play()` 메서드를 호출하여 SoundTransform 객체를 매개 변수로 전달합니다.

사운드가 재생되는 동안 `onEnterFrame()` 메서드가 반복적으로 실행됩니다. `onEnterFrame()` 메서드는 `Math.sin()` 함수를 사용하여 -1과 1 사이의 값을 생성합니다. 이 범위는 `SoundTransform.pan` 속성으로 사용할 수 있는 값에 해당합니다. SoundTransform 객체의 `pan` 속성은 새로운 값으로 설정되고 채널의 `soundTransform` 속성은 변경된 SoundTransform 객체를 사용하도록 설정됩니다.

이 예제를 실행하려면 bigSound.mp3라는 파일 이름을 로컬 mp3 파일 이름으로 대체합니다. 그리고 예제를 실행합니다. 오른쪽 채널 볼륨이 작아지면서 왼쪽 채널 볼륨이 커지고, 왼쪽 채널 볼륨이 작아지면서 오른쪽 채널 볼륨이 커집니다.

이 예제에서 SoundMixer 클래스의 `soundTransform` 속성을 설정하여 동일한 효과를 거둘 수 있습니다. 하지만 이 경우 이 SoundChannel 객체에 의해 재생되는 단일 사운드뿐 아니라 현재 재생되는 모든 사운드의 패닝에 영향을 미칩니다.

사운드 메타데이터를 사용한 작업

mp3 포맷을 사용하는 사운드 파일에는 해당 사운드에 대한 추가 데이터가 ID3 태그의 형태로 포함될 수 있습니다.

일부 mp3 파일에는 ID3 메타데이터가 없습니다. mp3 사운드 파일에 ID3 메타데이터가 포함된 경우, Sound 객체가 이 사운드 파일을 로드하면 `Event.ID3` 이벤트가 전달됩니다. 런타임 오류를 방지하려면 응용 프로그램에서 `Event.ID3` 이벤트를 수신할 때까지 기다린 후에 로드된 사운드의 `Sound.id3` 속성에 액세스해야 합니다.

다음 코드는 사운드 파일의 ID3 메타데이터가 로드된 시점을 인식하는 방법을 보여 줍니다.

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
  var id3:ID3Info = event.target.id3;

  trace("Received ID3 Info:");
  for (var propName:String in id3)
  {
    trace(propName + " = " + id3[propName]);
  }
}
```

이 코드는 Sound 객체를 만든 후 `Event.ID3` 이벤트를 수신하도록 지시합니다. 사운드 파일의 ID3 메타데이터가 로드되면 `onID3InfoReceived()` 메서드가 호출됩니다.

`onID3InfoReceived()` 메서드에 전달되는 Event 객체의 대상은 원본 Sound 객체이며, 이 메서드는 Sound 객체의 `id3` 속성을 가져온 다음, 이름이 지정된 모든 속성을 반복하여 그 값을 추적합니다.

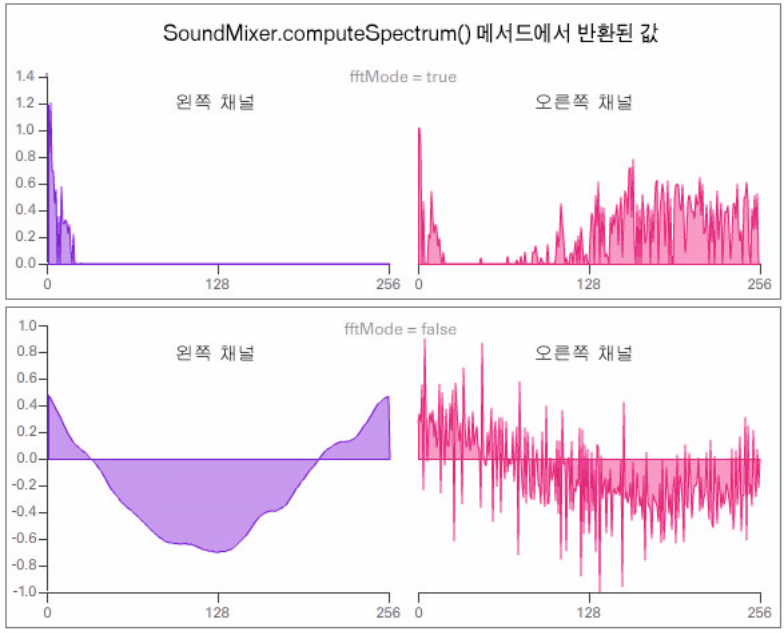
원시 사운드 데이터 액세스

SoundMixer.computeSpectrum() 메서드를 사용하면 현재 재생 중인 파형에 대한 원시 사운드 데이터를 응용 프로그램이 읽을 수 있습니다. 둘 이상의 SoundChannel 객체가 현재 재생 중인 경우, SoundMixer.computeSpectrum() 메서드는 함께 믹싱된 모든 SoundChannel 객체의 조합된 사운드 데이터를 표시합니다.

사운드 데이터는 512바이트의 데이터가 포함된 ByteArray 객체로 반환되며 각 개체는 -1과 1 사이의 부동 소수점 값을 포함합니다. 이러한 값은 재생 중인 사운드 파형의 위치 진폭을 나타냅니다. 값은 두 개의 256바이트 그룹으로 제공됩니다. 첫 번째 그룹은 왼쪽 스테레오 채널용이고 두 번째 그룹은 오른쪽 스테레오 채널용입니다.

FFTMode 매개 변수가 true로 설정된 경우, SoundMixer.computeSpectrum() 메서드는 파형 데이터가 아니라 주파수 스펙트럼 데이터를 반환합니다. 주파수 스펙트럼은 가장 낮은 주파수에서 가장 높은 주파수까지 사운드 주파수에 따라 배열된 진폭을 표시합니다. FFT(Fast Fourier Transform)는 파형 데이터를 주파수 스펙트럼 데이터로 변환하는 데 사용됩니다. 결과 주파수 스펙트럼 값은 0에서 약 1.414(2의 제곱근)까지입니다.

다음 그림에서는 FFTMode 매개 변수가 true로 설정되었을 때와 false로 설정되었을 때 computeSpectrum() 메서드로부터 반환된 데이터를 비교합니다. 이 그림에 사용된 사운드 데이터는 왼쪽 채널은 큰 베이스 사운드, 오른쪽 채널은 드럼 사운드에 대한 것입니다.



또한 computeSpectrum() 메서드는 더 낮은 비트율로 다시 샘플링된 데이터를 반환할 수 있습니다. 이렇게 하면 일반적으로 더 부드러운 파형 데이터 또는 주파수 데이터가 반환되고 세부적인 요소는 제거됩니다. stretchFactor 매개 변수는 computeSpectrum() 메서드 데이터가 샘플링되는 속도를 제어합니다. stretchFactor 매개 변수가 기본값인 0으로 설정된 경우 사운드 데이터는 44.1kHz의 속도로 샘플링됩니다. 이 속도는 stretchFactor 매개 변수의 각 연속 값에서 절반으로 되어 값 1은 22.05kHz, 값 2는 11.025kHz 등으로 속도를 지정합니다. computeSpectrum() 메서드는 더 높은 stretchFactor 값이 사용되어도 스테레오 채널당 256바이트를 반환합니다.

SoundMixer.computeSpectrum() 메서드는 다음과 같은 제한 사항이 있습니다.

- 마이크 또는 RTMP 스트림의 사운드 데이터는 전역 SoundMixer 객체를 통과하지 않으므로 SoundMixer.computeSpectrum() 메서드는 이러한 소스로부터 데이터를 반환하지 않습니다.
- 재생 중인 하나 이상의 사운드가 현재 내용 샌드박스 외부의 소스로부터 제공되는 경우 보안 제한 사항으로 인해 SoundMixer.computeSpectrum() 메서드에서 오류가 발생할 수 있습니다. SoundMixer.computeSpectrum() 메서드의 보안 제한에 대한 자세한 내용은 568페이지의 “사운드 로드 및 재생 시의 보안 고려 사항” 및 732페이지의 “데이터로 로드된 미디어 액세스”를 참조하십시오.

간단한 사운드 파형 표시기 만들기

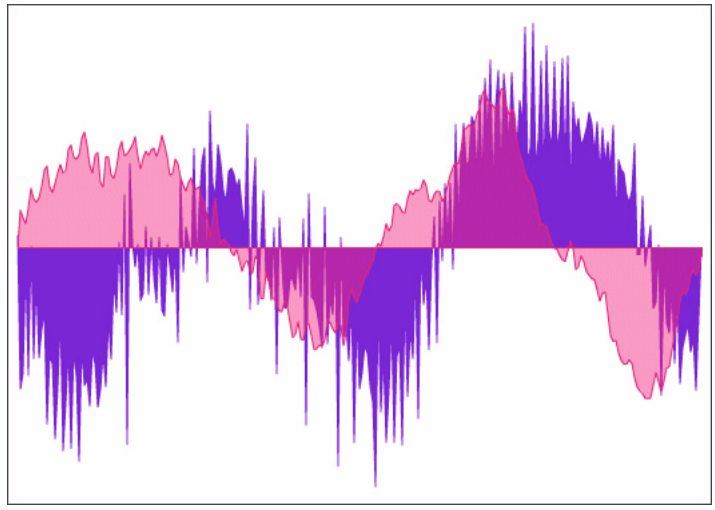
다음 예제에서는 SoundMixer.computeSpectrum() 메서드를 사용하여 각 프레임과 함께 움직이는 사운드 파형 차트를 표시합니다.

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();
```

```
function onEnterFrame(event:Event):void
{
  SoundMixer.computeSpectrum(bytes, false, 0);

  var g:Graphics = this.graphics;

  g.clear();
  g.lineStyle(0, 0x6600CC);
  g.beginFill(0x6600CC);
  g.moveTo(0, PLOT_HEIGHT);

  var n:Number = 0;

  // 왼쪽 채널
  for (var i:int = 0; i < CHANNEL_LENGTH; i++)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
  g.endFill();

  // 오른쪽 채널
  g.lineStyle(0, 0xCC0066);
  g.beginFill(0xCC0066, 0.5);
  g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

  for (i = CHANNEL_LENGTH; i > 0; i--)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(0, PLOT_HEIGHT);
  g.endFill();
}

function onPlaybackComplete(event:Event)
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

이 예제는 사운드 파일을 로드하여 재생한 다음, 사운드 재생 동안 `onEnterFrame()` 메서드를 트리거할 `Event.ENTER_FRAME` 이벤트를 수신합니다. `onEnterFrame()` 메서드는 사운드 웨이브 데이터를 `bytes` ByteArray 객체에 저장하는 `SoundMixer.computeSpectrum()` 메서드를 호출하는 것으로 시작됩니다.

사운드 파형은 벡터 드로잉 API를 사용하여 그려집니다. for 루프는 왼쪽 스테레오 채널을 나타내는 처음 256개 데이터 값을 찾은 후 Graphics.lineTo() 메서드를 사용하여 각각의 위치 사이에 차례로 선을 그립니다. 두 번째 for 루프는 다음 세트의 256개 값을 순환한 후 이 값들을 역순으로(오른쪽에서 왼쪽으로) 배치합니다. 결과 파형 형태는 다음 이미지와 같이 흥미로운 거울 이미지 효과를 나타낼 수 있습니다.



사운드 입력 캡처

Microphone 클래스를 사용하면 응용 프로그램이 사용자 시스템의 마이크 또는 기타 사운드 입력 장치에 연결하여 입력 오디오를 해당 시스템에 브로드캐스팅하거나 오디오 데이터를 Flash Media Server 등의 원격 서버에 보냅니다.

마이크 액세스

Microphone 클래스에는 생성자 메서드가 없습니다. 대신 다음과 같이 정적 Microphone.getMicrophone() 메서드를 사용하여 새 Microphone 인스턴스를 구합니다.

```
var mic:Microphone = Microphone.getMicrophone();
```

매개 변수 없이 Microphone.getMicrophone() 메서드를 호출하면 사용자 시스템에서 검색된 첫 번째 사운드 입력 장치가 반환됩니다.

시스템에는 둘 이상의 사운드 입력 장치가 연결되어 있을 수 있습니다. 응용 프로그램은 Microphone.names 속성을 사용하여 사용 가능한 모든 사운드 입력 장치의 이름 배열을 가져옵니다. 그런 다음 배열에 있는 장치 이름의 인덱스 값과 일치하는 index 매개 변수를 사용하여 Microphone.getMicrophone() 메서드를 호출합니다.

시스템에 마이크 또는 기타 사운드 입력 장치가 연결되어 있지 않을 수도 있습니다. Microphone.names 속성 또는 Microphone.getMicrophone() 메서드를 사용하여 사용자의 시스템에 사운드 입력 장치가 설치되어 있는지 여부를 확인합니다. 사용자의 시스템에 사운드 입력 장치가 설치되어 있지 않으면 names 배열 길이는 0이고 getMicrophone() 메서드가 null 값을 반환합니다.

응용 프로그램이 Microphone.getMicrophone() 메서드를 호출하면 Flash Player는 [Flash Player 설정] 대화 상자를 표시합니다. 이 대화 상자에서 사용자는 시스템의 카메라 및 마이크에 대한 Flash Player의 액세스를 허용 또는 거부할 수 있습니다. 이 대화 상자에서 [허용] 또는 [거부] 버튼을 클릭하면 StatusEvent가 전달됩니다. 다음 예제에서와 같이, 해당 StatusEvent 인스턴스의 code 속성은 마이크 액세스의 허용 또는 거부를 나타냅니다.

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
  if (event.code == "Microphone.Unmuted")
  {
    trace("Microphone access was allowed.");
  }
  else if (event.code == "Microphone.Muted")
  (
    trace("Microphone access was denied.");
  }
}
```

StatusEvent.code 속성에는 “Microphone.Unmuted”(액세스가 허용된 경우) 또는 “Microphone.Muted”(액세스가 거부된 경우)가 포함됩니다.

예외

사용자가 마이크 액세스를 허용 또는 거부할 때 Microphone.muted 속성은 각각 true 또는 false로 설정됩니다. 하지만 muted 속성은 StatusEvent가 전달될 때까지 Microphone 인스턴스에 설정되지 않으므로 응용 프로그램은 Microphone.muted 속성을 확인하기 전에 StatusEvent.STATUS 이벤트가 전달되기를 기다려야 합니다.

마이크 오디오를 로컬 스피커로 라우팅

매개 변수 값을 `true`로 하여 `Microphone.setLoopback()` 메서드를 호출하면 마이크의 오디오 입력을 로컬 시스템 스피커로 라우팅할 수 있습니다.

로컬 마이크의 사운드가 로컬 스피커로 라우팅되면 오디오 피드백 루프를 만들어 커다란 빽 소리를 일으키고 잠재적으로 사운드 하드웨어의 손상을 초래할 위험이 있습니다. 매개 변수 값을 `true`로 하여 `Microphone.setUseEchoSuppression()` 메서드를 호출하면 오디오 피드백이 발생할 위험이 완전히 배제되지는 않지만 낮아집니다. 사용자가 헤드폰을 사용하거나 스피커 이외의 다른 장치를 사용하여 사운드를 재생하고 있음을 확신할 수 없는 경우라면 `Microphone.setLoopback(true)`을 호출하기 전에 항상 `Microphone.setUseEchoSuppression(true)`을 호출하는 것이 좋습니다.

다음 코드는 로컬 마이크의 오디오를 로컬 시스템 스피커로 라우팅하는 방법을 보여 줍니다.

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

마이크 오디오 변경

응용 프로그램은 마이크로부터 전달되는 오디오 데이터를 두 가지 방법으로 변경할 수 있습니다. 먼저 입력 사운드의 게인을 변경할 수 있습니다. 게인은 입력 값을 지정된 양만큼 곱하여 더 크거나 작은 사운드를 생성합니다. `Microphone.gain` 속성은 0부터 100까지의 숫자 값을 받습니다. 값을 50으로 지정하면 1을 곱한 것과 같게 되어 보통 볼륨을 생성합니다. 값을 0으로 지정하면 0을 곱한 것과 같게 되어 입력 오디오를 제거합니다. 값을 50 이상으로 지정하면 보통 볼륨보다 높은 볼륨을 생성합니다.

응용 프로그램에서 입력 오디오의 샘플링 속도를 변경할 수도 있습니다. 샘플링 속도가 높을수록 사운드 품질은 향상되지만, 전송과 저장에 더 많은 리소스를 사용하는 좀 더 밀집된 데이터 스트림이 생성됩니다. `Microphone.rate` 속성은 kHz로 측정되는 오디오 샘플링 속도를 나타냅니다. 기본 샘플링 속도는 8kHz입니다. 마이크가 더 높은 속도를 지원하는 경우 `Microphone.rate` 속성 값을 8kHz 이상으로 설정할 수 있습니다. 예를 들어 `Microphone.rate` 속성의 값을 11로 설정하면 샘플링 속도가 11kHz로 지정되고, 22로 설정하면 샘플링 속도가 22kHz로 지정됩니다.

마이크 활동 감지

대역폭과 처리 리소스를 보존하기 위해 Flash Player는 마이크가 사운드를 전송하지 않는 때를 감지하려고 합니다. 마이크의 활동 레벨이 일정 시간 동안 묵음 레벨 임계값 미만을 유지하면 Flash Player는 오디오 입력 전송을 중지하고 대신 단순 ActivityEvent를 전달합니다.

Microphone 클래스의 다음 세 가지 속성은 활동을 모니터링하고 제어합니다.

- 읽기 전용 `activityLevel` 속성은 마이크가 감지하는 사운드의 양을 0~100의 등급으로 나타냅니다.
- `silenceLevel` 속성은 마이크 활성화에 필요한 사운드의 양을 지정하고 `ActivityEvent.ACTIVITY` 이벤트를 전달합니다. `silenceLevel` 속성도 0~100의 등급을 사용하고 기본값은 10입니다.
- `silenceTimeout` 속성은 마이크가 현재 사용 중이지 않음을 알리기 위해 `ActivityEvent.ACTIVITY` 이벤트가 전달될 때까지 활동 레벨이 묵음 수준 미만을 유지해야 하는 시간을 밀리초 단위로 정의합니다. 기본 `silenceTimeout` 값은 2000입니다.

`Microphone.silenceLevel` 속성과 `Microphone.silenceTimeout` 속성은 읽기 전용이지만 `Microphone.setSilenceLevel()` 메서드를 사용하여 이 값을 변경할 수 있습니다.

새로운 활동이 감지되었을 때 마이크를 활성화하는 프로세스에서 약간의 지연이 발생하는 경우도 있습니다. 마이크를 항상 활성화시켜 두면 이러한 활성화 지연을 방지할 수 있습니다. 응용 프로그램이 `silenceLevel` 매개 변수를 0으로 설정하여 `Microphone.setSilenceLevel()` 메서드를 호출하면 Flash Player는 사운드가 감지되지 않더라도 마이크의 활성 상태를 유지하고 오디오 데이터를 계속 수집합니다. 반대로, `silenceLevel` 매개 변수를 100으로 설정하면 마이크가 전혀 활성화되지 않습니다.

다음 예제는 마이크에 대한 정보를 표시하고 Microphone 객체가 전달한 activity 이벤트 및 status 이벤트를 보고합니다.

```
import flash,events.ActivityEvent;
import flash,events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
  trace("    " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);
```

```
mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
  trace("activating=" + event.activating + ", activityLevel=" +
    mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
  trace("status: level=" + event.level + ", code=" + event.code);
}
```

위 예제를 실행할 때 시스템 마이크에 대고 말하거나 소리를 낸 다음, 콘솔이나 디버그 윈도우에 표시되는 결과 trace 문을 확인합니다.

미디어 서버에서의 오디오 송수신

Flash Media Server와 같은 스트리밍 미디어 서버와 함께 ActionScript를 사용할 때 추가 오디오 기능을 사용할 수 있습니다.

특히 응용 프로그램은 Microphone 객체를 NetStream 객체에 연결하여 마이크에서 서버로 직접 데이터를 전송할 수 있습니다. 또한 오디오 데이터는 서버에서 Flash 또는 Flex 응용 프로그램으로 스트리밍되어 MovieClip의 일부로서 또는 Video 객체를 사용하여 재생될 수 있습니다.

자세한 내용은 http://livedocs.macromedia.com_kr에서 온라인으로 제공되는 Flash Media Server 설명서를 참조하십시오.

예제: Podcast Player

포드캐스트는 인터넷이나 주문 또는 구독을 통해 배포되는 사운드 파일입니다. 일반적으로 포드캐스트는 포드캐스트 채널이라고도 하는 시리즈의 일부로 제작됩니다. 포드캐스트 에피소드는 길이가 1분이 될 수도 있고 몇 시간이 될 수도 있으므로 일반적으로 스트리밍으로 재생됩니다. 포드캐스트 에피소드는 아이템이라고도 하며 보통 mp3 파일 포맷으로 전달됩니다. 비디오 포드캐스트도 널리 사용되지만 이 샘플 응용 프로그램은 mp3 파일을 사용하는 오디오 포드캐스트만 재생합니다.

이 예제는 모든 기능을 갖춘 포드캐스트 수집기 응용 프로그램이 아닙니다. 예를 들어, 특정 포드캐스트에 대한 구독을 관리하지 않으며 사용자가 이전에 들은 포드캐스트가 다음에 응용 프로그램이 실행될 때 기억되지 않습니다. 이 예제는 더 복잡한 기능의 포드캐스트 수집기 작성을 위한 토대가 될 수 있습니다.

Podcast Player 예제는 다음과 같은 ActionScript 프로그래밍 기술을 보여 줍니다.

- 외부 RSS 피드 읽기 및 해당 XML 내용 파싱
- SoundFacade 클래스를 만들어 사운드 파일을 간편하게 로드 및 재생
- 사운드 재생 진행률 표시
- 사운드 재생 일시 정지 및 다시 시작

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. Podcast Player 응용 프로그램 파일은 Samples/PodcastPlayer 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
PodcastPlayer.mxml 또는 PodcastPlayer.fla	Flex(MXML) 또는 Flash(FLA)용 응용 프로그램의 사용자 인터페이스입니다.
RSSBase.as	RSSChannel 클래스와 RSSItem 클래스에 대한 일반 속성 및 메서드를 제공하는 기본 클래스입니다.
RSSChannel.as	RSS 채널에 대한 데이터를 보관하는 ActionScript 클래스입니다.
RSSItem.as	RSS 항목에 대한 데이터를 보관하는 ActionScript 클래스입니다.
SoundFacade.as	응용 프로그램에 대한 기본 ActionScript 클래스입니다. Sound 클래스와 SoundChannel 클래스의 메서드 및 이벤트를 캡슐화하고 재생 일시 정지 및 다시 시작에 대한 지원을 추가합니다.

파일	설명
URLService.as	원격 URL로부터 데이터를 가져오는 ActionScript 클래스입니다.
playerconfig.xml	포드캐스트 채널을 나타내는 RSS 피드 목록이 포함된 XML 파일입니다.

포드캐스트 채널용 RSS 데이터 읽기

Podcast Player 응용 프로그램은 몇 개의 포드캐스트 채널과 해당 에피소드에 대한 정보를 읽는 것으로 시작합니다.

1. 먼저 응용 프로그램은 포드캐스트 채널 목록이 포함된 XML 구성 파일을 읽고 채널 목록을 사용자에게 표시합니다.

2. 사용자가 포드캐스트 채널 중 하나를 선택하면 응용 프로그램은 채널에 대한 RSS 피드를 읽고 채널 에피소드 목록을 표시합니다.

이 예제는 URLLoader 유틸리티 클래스를 사용하여 원격 위치 또는 로컬 파일로부터 텍스트 기반 데이터를 가져옵니다. Podcast Player는 먼저 playerconfig.xml 파일로부터 XML 포맷의 RSS 피드 목록을 가져오기 위한 URLLoader 객체를 만듭니다. 그리고 사용자가 목록에서 특정 피드를 선택하면 새로운 URLLoader 객체가 만들어져 해당 피드의 URL로부터 RSS 데이터를 읽습니다.

SoundFacade 클래스를 사용하여 사운드 로드 및 재생 단순화

ActionScript 3.0 사운드 아키텍처는 강력하지만 복잡합니다. 기본적인 사운드 로드와 재생 기능만 필요한 응용 프로그램은 간단한 메서드 호출 및 이벤트 집합을 제공하여 일부 복잡한 기능을 숨기는 클래스를 사용할 수 있습니다. 이러한 클래스는 소프트웨어 디자인 패턴 분야에서 *facade*라고 합니다.

SoundFacade 클래스는 다음과 같은 작업을 수행하는 단일 인터페이스를 제공합니다.

- Sound 객체, SoundLoaderContext 객체 및 SoundMixer 클래스를 사용하여 사운드 파일 로드
- Sound 객체 및 SoundChannel 객체를 사용하여 사운드 파일 재생
- 재생 진행률 이벤트 전달
- Sound 객체 및 SoundChannel 객체를 사용하여 사운드 재생 일시 정지 및 다시 시작

SoundFacade 클래스는 대부분의 ActionScript 사운드 클래스 기능을 보다 단순하게 제공하려고 합니다.

다음 코드는 클래스 선언, 클래스 속성 및 SoundFacade() 생성자 메서드를 보여 줍니다.

```
public class SoundFacade extends EventDispatcher
{
  public var s:Sound;
  public var sc:SoundChannel;
  public var url:String;
  public var bufferTime:int = 1000;

  public var isLoaded:Boolean = false;
  public var isReadyToPlay:Boolean = false;
  public var isPlaying:Boolean = false;
  public var isStreaming:Boolean = true;
  public var autoLoad:Boolean = true;
  public var autoPlay:Boolean = true;

  public var pausePosition:int = 0;

  public static const PLAY_PROGRESS:String = "playProgress";
  public var progressInterval:int = 1000;
  public var playTimer:Timer;

  public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                    autoPlay:Boolean = true, streaming:Boolean = true,
                    bufferTime:int = -1):void
  {
    this.url = soundUrl;

    // 이 객체의 비헤이비어를 결정하는 Boolean 값을 설정합니다 .
    this.autoLoad = autoLoad;
    this.autoPlay = autoPlay;
    this.isStreaming = streaming;

    // 전역 bufferTime 값이 기본값으로 사용됩니다 .
    if (bufferTime < 0)
    {
      bufferTime = SoundMixer.bufferTime;
    }

    // 0-30 초의 적절한 버퍼링 시간을 유지합니다 .


    this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

    if (autoLoad)
    {
      load();
    }
  }
```

SoundFacade 클래스는 EventDispatcher 클래스를 확장하여 자체의 이벤트를 전달하도록 합니다. 이 클래스 코드는 먼저 Sound 객체와 SoundChannel 객체의 속성을 선언합니다. 이 클래스는 사운드를 스트리밍할 때 사용할 bufferTime 속성과 사운드 파일의 URL 값도 저장합니다. 또한 로딩 및 재생 비헤이비어에 영향을 미치는 다음과 같은 몇 가지 부울 매개 변수 값을 받습니다.

- autoLoad 매개 변수는 객체가 만들어지면 바로 사운드 로드를 시작하도록 객체에 지시합니다.
- autoPlay 매개 변수는 충분한 사운드 데이터가 로드되면 바로 재생을 시작함을 나타냅니다. 스트리밍 사운드의 경우에는 bufferTime 속성에 지정된 충분한 데이터가 로드되었을 때 재생이 시작됩니다.
- streaming 매개 변수는 로드가 완료되기 전에 이 사운드 파일 재생을 시작할 수 있음을 나타냅니다.

bufferTime 매개 변수의 기본값은 -1입니다. 생성자 메서드가 bufferTime 매개 변수에서 음수 값을 감지하면 bufferTime 속성이 SoundMixer.bufferTime 값으로 설정됩니다. 이렇게 되면 필요한 전역 SoundMixer.bufferTime 값으로 응용 프로그램의 기본값이 설정될 수 있습니다.

autoLoad 매개 변수가 true로 설정된 경우, 생성자 메서드는 즉시 다음 load() 메서드를 호출하여 사운드 파일 로드를 시작합니다.

```
public function load():void
{
  if (this.isPlaying)
  {
    this.stop();
    this.s.close();
  }
  this.isLoaded = false;

  this.s = new Sound();

  this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
  this.s.addEventListener(Event.OPEN, onLoadOpen);
  this.s.addEventListener(Event.COMPLETE, onLoadComplete);
  this.s.addEventListener(Event.ID3, onID3);
  this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
  this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

  var req:URLRequest = new URLRequest(this.url);

  var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime,
                                                          true);
  this.s.load(req, this.slc);
}
```

load() 메서드는 새로운 Sound 객체를 생성한 후 모든 중요 사운드 이벤트에 대한 리스너를 추가합니다. 그런 다음 Sound 객체에 사운드 파일 로드를 지시하고 SoundLoaderContext 객체를 사용하여 bufferTime 값을 전달하도록 합니다.

url 속성은 변경 가능하므로, SoundFacade 인스턴스를 사용하여 여러 사운드 파일을 연속적으로 재생할 수 있습니다. url 속성을 변경하고 load() 메서드를 호출하기만 하면 새 사운드 파일이 로드됩니다.

다음 세 가지 이벤트 리스너 메서드는 SoundFacade 객체가 로드 프로세스를 추적하고 사운드 재생 시점을 결정하는 방법을 보여 줍니다.

```
public function onLoadOpen(event:Event):void
{
  if (this.isStreaming)
  {
    this.isReadyToPlay = true;
    if (autoPlay)
    {
      this.play();
    }
  }
  this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
  this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
  this.isReadyToPlay = true;
  this.isLoaded = true;
  this.dispatchEvent(evt.clone());

  if (autoPlay && !isPlaying)
  {
    play();
  }
}
```

사운드 로드가 시작되면 onLoadOpen() 메서드가 실행됩니다. 사운드가 스트리밍 모드에서 재생될 수 있는 경우 onLoadComplete() 메서드는 즉시 isReadyToPlay 플래그를 true로 설정합니다. isReadyToPlay 플래그는 사용자가 재생 버튼을 클릭하는 등의 액션을 수행할 때 응용 프로그램이 사운드 재생을 시작할 수 있는지 여부를 결정합니다. SoundChannel 클래스는 사운드 데이터의 버퍼링을 관리하므로 play() 메서드를 호출하기 전에 충분한 데이터가 로드되었는지 명시적으로 확인할 필요가 없습니다.

onLoadProgress() 메서드는 로드 프로세스 중에 정기적으로 실행됩니다. 이 메서드는 이 SoundFacade 객체를 사용하는 코드가 사용할 해당 ProgressEvent 객체의 복제본을 전달합니다.

사운드 데이터가 완전히 로드되었으면 onLoadComplete() 메서드가 실행되어, 필요한 경우 비스트리밍 사운드에 대해 play() 메서드를 호출합니다. play() 메서드는 아래와 같습니다.

```
public function play(pos:int = 0):void
{
  if (!this.isPlaying)
  {
    if (this.isReadyToPlay)
    {
      this.sc = this.s.play(pos);
      this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
      this.isPlaying = true;

      this.playTimer = new Timer(this.progressInterval);
      this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
      this.playTimer.start();
    }
  }
}
```

사운드 재생 준비가 되었으면 play() 메서드가 Sound.play() 메서드를 호출합니다. 결과 SoundChannel 객체는 sc 속성에 저장됩니다. 그런 다음 play() 메서드는 재생 진행률 이벤트를 정기적으로 전달하는 데 사용될 Timer 객체를 만듭니다.

재생 진행률 표시

재생 모니터링을 구동하는 Timer 객체 생성은 복잡한 작업으로 한 번만 코딩해야 합니다. 이 Timer 논리를 SoundFacade 클래스 등의 다시 사용할 수 있는 클래스에 캡슐화하면 사운드 로드와 재생 시 동일한 종류의 진행률 이벤트를 응용 프로그램이 수신할 수 있습니다.

SoundFacade.play() 메서드가 생성한 Timer 객체는 매 초마다 TimerEvent 인스턴스를 전달합니다. 다음 onPlayTimer() 메서드는 새 TimerEvent가 도착할 때마다 실행됩니다.

```
public function onPlayTimer(event:TimerEvent):void
{
  var estimatedLength:int =
    Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
  var progEvent:ProgressEvent =
    new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position,
  estimatedLength);
  this.dispatchEvent(progEvent);
}
```

onPlayTimer() 메서드는 566페이지의 "재생 모니터링" 단원에 설명된 크기 예측 기법을 구현합니다. 그런 다음, 이벤트 유형이 SoundFacade.PLAY_PROGRESS bytesLoaded 속성이 SoundChannel 객체의 현재 위치로 설정되고 bytesTotal 속성이 사운드 데이터의 추정된 길이로 설정된 새로운 ProgressEvent 인스턴스를 만듭니다.

재생 일시 정지 및 다시 시작

앞에서 설명한 SoundFacade.play() 메서드는 사운드 데이터의 시작 위치에 해당하는 pos 매개 변수를 받습니다. pos 값이 0이면 사운드는 처음부터 재생을 시작합니다.

아래와 같이 SoundFacade.stop() 메서드도 pos 매개 변수를 받습니다.

```
public function stop(pos:int = 0):void
{
  if (this.isPlaying)
  {
    this.pausePosition = pos;
    this.sc.stop();
    this.playTimer.stop();
    this.isPlaying = false;
  }
}
```

SoundFacade.stop() 메서드가 호출될 때마다 동일한 사운드의 재생을 사용자가 다시 시작하려는 경우 응용 프로그램이 재생 헤드의 위치를 지정할 수 있도록 pausePosition 속성이 설정됩니다.

아래와 같은 SoundFacade.pause() 및 SoundFacade.resume() 메서드는 SoundFacade.stop() 및 SoundFacade.play() 메서드를 각각 호출하여 pos 매개 변수 값을 전달합니다.

```
public function pause():void
{
  stop(this.sc.position);
}

public function resume():void
{
  play(this.pausePosition);
}
```

pause() 메서드는 현재의 SoundChannel.position 값을 play() 메서드에 전달하고 play() 메서드는 해당 값을 pausePosition 속성에 저장합니다. resume() 메서드는 pausePosition 값을 시작 위치로 사용하여 동일한 사운드의 재생을 다시 시작합니다.

Podcast Player 예제 확장

이 예제에서는 재사용 가능한 SoundFacade 클래스의 기능을 보여 주는 Podcast Player의 핵심을 나타냅니다. 이 응용 프로그램의 기능을 확장하기 위해 다음과 같은 기타 기능을 추가할 수 있습니다.

- 사용자가 응용 프로그램을 다음에 실행할 때 사용할 수 있는 SharedObject 인스턴스에 각 에피소드와 관련한 피드 및 사용 정보 목록을 저장합니다.
- 사용자가 자신의 RSS 피드를 포드캐스트 채널 목록에 추가할 수 있도록 합니다.
- 사용자가 에피소드를 중지하거나 종료했을 때 재생 헤드의 위치를 기억하여 사용자가 다음에 응용 프로그램을 실행할 때 해당 위치에서 다시 시작할 수 있도록 합니다.
- 사용자가 인터넷에 연결되어 있지 않을 때 오프라인 상태에서 수신하도록 에피소드 mp3 파일을 다운로드합니다.
- 포드캐스트 채널에서 새로운 에피소드가 있는지 정기적으로 확인하고 에피소드 목록을 자동으로 업데이트하는 구독 기능을 추가합니다.
- Odeo.com 등의 포드캐스트 호스팅 서비스로부터 API를 사용하여 포드캐스트 검색 및 탐색 기능을 추가합니다.

제 21 장

사용자 입력 캡처

21

이 장에서는 ActionScript 3.0을 사용하여 대화형 작업을 만들어서 사용자 작업에 응답하는 방법을 설명합니다. 키보드와 마우스 이벤트에 대해 설명하고 컨텍스트 메뉴 및 포커스 관리 사용자 정의를 비롯한 고급 기능에 대해서도 설명합니다. 마지막으로 마우스 입력에 응답하는 응용 프로그램의 예인 WordSearch를 소개하는 것으로 이 장을 마무리합니다.

이 장의 내용은 사용자가 ActionScript 3.0 이벤트 모델 사용에 익숙함을 전제로 하여 설명합니다. 자세한 내용은 295페이지의 제10장, “이벤트 처리”를 참조하십시오.

목차

사용자 입력에 대한 기본적인 사항

사용자 입력 캡처 소개

키보드, 마우스, 카메라 또는 이러한 장치의 조합을 통한 사용자 상호 작용은 가장 기본적인 상호 작용입니다. ActionScript 3.0에서 사용자 상호 작용에 대한 식별 및 응답은 주로 이벤트 수신과 관련되어 있습니다.

DisplayObject 클래스의 하위 클래스인 InteractiveObject 클래스는 사용자 상호 작용을 처리하는 데 필요한 일반적인 기능과 이벤트 구조를 제공합니다. InteractiveObject 클래스의 인스턴스를 직접 만들지는 않습니다. 대신 SimpleButton, Sprite, TextField와 여러 Flash 및 Flex 구성 요소 등의 표시 객체는 이 클래스로부터 사용자 상호 작용 모델을 상속하므로 일반 구조를 공유합니다. 따라서 InteractiveObject로부터 파생된 객체에서 사용자 상호 작용을 처리하기 위해 작성하는 코드와 배운 기술은 다른 모든 분야에도 적용이 가능합니다.

이 장에서는 다음과 같은 일반적인 사용자 상호 작용 작업에 대해 설명합니다.

- 응용 프로그램 전반의 키보드 입력 캡처
- 특정 표시 객체에 대한 키보드 입력 캡처
- 응용 프로그램 전반의 마우스 액션 캡처
- 특정 표시 객체에 대한 마우스 입력 캡처
- 드래그 앤 드롭 상호 작용 만들기
- 사용자 정의 마우스 커서(마우스 포인터) 만들기
- 컨텍스트 메뉴에 신규 비헤이비어 추가
- 포커스 관리

중요한 개념 및 용어

계속하기 전에, 사용자 상호 작용에 관한 다음의 주요 용어를 익혀야 합니다.

- 문자 코드: 키보드에서 누르는 키와 관련하여 현재 문자 집합에서의 문자를 나타내는 숫자 코드입니다. 예를 들어, "D"와 "d"는 문자 코드는 서로 다르지만 영문 키보드에서 동일한 키로 입력됩니다.
- 컨텍스트 메뉴: 사용자가 마우스 오른쪽 버튼을 클릭하거나 특정 키보드-마우스 조합을 사용할 때 나타나는 메뉴입니다. 컨텍스트 메뉴 명령은 일반적으로 클릭한 대상에 직접 적용되는 경우가 많습니다. 예를 들어, 이미지에 대한 컨텍스트 메뉴에는 이미지를 별도의 창으로 표시하는 명령과 이미지를 다운로드하는 명령이 포함됩니다.
- 포커스: 선택한 요소가 활성화되어 있으며 키보드 또는 마우스 상호 작용의 대상이 됨을 나타냅니다.
- 키 코드: 키보드의 실제 키에 해당되는 숫자 코드입니다.

이 장의 예제를 사용하여 작업

이 장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장에서는 ActionScript의 사용자 입력에 대해 다룹니다. 따라서 기본적으로 이 장의 모든 코드 샘플은 주로 텍스트 필드나 InteractiveObject 하위 클래스 등 일종의 표시 객체를 조작합니다. 예제의 목적에 따라 Adobe Flash CS3 Professional의 스테이지에서 만들고 배치한 표시 객체 또는 ActionScript로 만든 객체를 사용하기도 합니다. Flash Player에서 결과를 확인하고 샘플을 조작하면서 코드의 효과를 알아보는 것도 샘플 테스트에 필요한 과정입니다.

이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. 스테이지에서 인스턴스를 만듭니다.

- 텍스트 필드를 참조하는 코드라면 [텍스트 도구]를 사용하여 스테이지에서 새 동적 텍스트 필드를 만듭니다.
- 또는 스테이지에서 버튼이나 무비 클립 심볼 인스턴스를 만듭니다.

5. 만들어진 텍스트 필드, 버튼 또는 무비 클립을 선택하고 속성 관리자에서 인스턴스 이름을 지정합니다. 이름은 샘플 코드의 표시 객체 이름과 일치해야 합니다. 예를 들어, `myDisplayObject`라는 객체를 조작하는 코드라면 스테이지 객체 이름도 `myDisplayObject`가 되어야 합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   해당 코드에 지정된 대로 화면의 표시 객체가 조작됩니다.

키보드 입력 캡처

InteractiveObject 클래스에서 상호 작용 모델을 상속한 표시 객체는 이벤트 리스너를 사용하여 키보드 이벤트에 응답할 수 있습니다. 예를 들어, Stage에 이벤트 리스너를 배치하여 키보드 입력을 수신하고 이에 응답할 수 있습니다. 다음 코드에서는 이벤트 리스너가 키보드 입력을 캡처하고 키 이름과 키 코드 속성이 표시됩니다.

```
function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (character code: " + event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Ctrl 키와 같은 몇몇 키는 글리프 표현 없이도 이벤트를 생성합니다.

이전 코드 예제에서 키보드 이벤트 리스너는 전체 Stage에 대한 키보드 입력을 캡처했습니다. Stage의 특정 표시 객체에 대한 이벤트 리스너를 작성할 수도 있습니다. 이 이벤트 리스너는 객체에 포커스가 맞춰지면 트리거됩니다.

다음 예제에서는 사용자가 TextField 인스턴스 내에 입력할 때만 입력한 키가 출력 패널에 반영됩니다. Shift 키를 누르고 있으면 TextField의 테두리 색상이 일시적으로 빨간색으로 바뀝니다.

이 코드는 스테이지에 `tf`라는 이름의 TextField 인스턴스가 있다고 가정합니다.

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (Key code: " + event.keyCode + " character code: "
    + event.charCode + ")");
```

```
  if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
  trace("Key Released: " + String.fromCharCode(event.charCode) +
    " (Key code: " + event.keyCode + " character code: " +
    event.charCode + ")");
  if (event.keyCode == Keyboard.SHIFT)
  {
    tf.borderColor = 0x000000;
  }
}
```

또한 TextField 클래스는 `textInput` 이벤트를 보고하며, 사용자는 텍스트를 입력할 때 이를 수신할 수 있습니다. 자세한 내용은 486페이지의 "텍스트 입력 캡처"를 참조하십시오.

키 코드 및 문자 코드 이해

키보드 이벤트의 `keyCode` 및 `charCode` 속성에 액세스하여 어떤 키를 눌렀는지 확인하고 다른 액션을 트리거할 수 있습니다. `keyCode` 속성은 키보드의 특정 키 값에 해당하는 숫자 값이고, `charCode` 속성은 현재의 문자 세트에서 해당 키가 가지는 숫자 값입니다. 이때 기본 문자 집합은 ASCII를 지원하는 UTF-8입니다.

키 코드와 문자 값에는 주요한 차이가 있는데, 키 코드 값은 키보드의 특정 키를 나타내고(키패드의 1 키는 키보드 상단의 1 키와 다르지만 "1"을 입력하는 키와 "!"를 입력하는 키는 동일), 문자 값은 특정 문자를 나타냅니다(R 문자와 r 문자는 서로 다름).

예제 **특정 키와 해당 키의 문자 코드 값(ASCII) 사이의 매핑을 보려면 685페이지의 부 록 C, "키보드 키 및 키 코드 값"을 참조하십시오.**

키와 키 코드의 매핑은 장치 및 운영 체제에 따라 다릅니다. 따라서 키 매핑을 사용하여 액션을 트리거해서는 안 됩니다. 대신에 Keyboard 클래스에서 제공하는 미리 정의된 상수 값을 사용하여 적당한 `keyCode` 속성을 참조해야 합니다. 예를 들어, Shift 키의 키 매핑 대신 `Keyboard.SHIFT` 상수를 사용하십시오(위의 코드 샘플 참조).

KeyboardEvent 우선 순위 이해

다른 이벤트와 마찬가지로 키보드 이벤트 시퀀스도 `addEventListener()` 메서드의 코드 내 지정 순서가 아닌 표시 객체의 계층 구조에 의해 결정됩니다.

예를 들어, 다음 예제와 같이 `container`라는 무비 클립 내에 `tf` 텍스트 필드를 배치하고 두 인스턴스에 키보드 이벤트의 이벤트 리스너를 추가하는 경우가 있을 수 있습니다.

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
```

```
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
  trace(event.currentTarget.name + " hears key press: " +
    String.fromCharCode(event.charCode) + " (key code: " +
    event.keyCode + " character code: " + event.charCode + ")");
}
```

텍스트 필드와 부모 컨테이너에 모두 리스너가 있기 때문에 텍스트 필드에 키를 입력할 때마다 `reportKeyDown()` 함수가 두 번씩 호출됩니다. 키를 하나 누르면 텍스트 필드가 이벤트를 전달하고 그 뒤에 `container` 무비 클립이 이벤트를 전달합니다.

운영 체제와 웹 브라우저는 Adobe Flash Player에 앞서 키보드 이벤트를 처리합니다. 예를 들어, Microsoft Internet Explorer에서 Ctrl+W를 누르면 포함된 SWF 파일이 키보드 이벤트를 전달하기 전에 브라우저 창이 닫힙니다.

마우스 입력 캡처

마우스를 클릭하면 대화형 기능을 트리거하는 데 사용할 수 있는 마우스 이벤트가 만들어집니다. 이벤트 리스너를 Stage에 추가하면 SWF 내에서 발생하는 마우스 이벤트를 수신할 수 있습니다. 또한 InteractiveObject로부터 상속되는 Stage의 객체에(예: Sprite 또는 MovieClip) 이벤트 리스너를 추가할 수 있습니다. 이러한 리스너는 객체를 클릭하면 트리거됩니다.

키보드 이벤트와 마찬가지로 마우스 이벤트도 버블링됩니다. 다음 예제에서 `square`는 Stage의 자식이므로 정사각형을 클릭하면 Sprite `square`와 Stage 객체 둘 다에서 이벤트가 전달됩니다.

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
  trace(event.currentTarget.toString() +
    " dispatches MouseEvent. Local coords [" +
    event.localX + "," + event.localY + "] Stage coords [" +
    event.stageX + "," + event.stageY + "]");
}
```

위 예제의 마우스 이벤트에는 클릭 위치 정보가 포함되어 있습니다. localX 및 localY 속성에는 표시 체인의 최하위 자식에 대한 마우스 클릭 위치가 포함되어 있습니다. 예를 들어, square의 왼쪽 위 모서리를 클릭하면 square의 등록 포인트인 로컬 좌표 [0,0]이 보고됩니다. 또는 stageX 및 stageY 속성이 스테이지 클릭의 전역 좌표를 참조합니다. 이 좌표의 경우 같은 클릭으로 [50,50]이 보고됩니다. square가 해당 좌표로 이동되었기 때문입니다. 이러한 두 좌표 쌍은 사용자 상호 작용에 대한 응답 방식에 따라 유용할 수 있습니다.

MouseEvent 객체에는 altKey, ctrlKey 및 shiftKey라는 부울 속성도 포함되어 있습니다. 이러한 속성을 사용하여 마우스 클릭 시 Alt, Ctrl 또는 Shift 키를 함께 눌렀는지 여부를 확인할 수 있습니다.

드래그 앤 드롭 기능 만들기

드래그 앤 드롭 기능을 사용하면 마우스 왼쪽 버튼을 누른 상태에서 객체를 드래그하여 화면의 새로운 위치로 이동할 수 있습니다. 아래의 코드는 여기에 해당하는 예제입니다.

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
  circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
```

```
  circle.stopDrag();
  trace(circle.dropTarget.name);
}
```

자세한 내용은 373페이지의 “드래그 앤 드롭 상호 작용 만들기”를 참조하십시오.

마우스 커서 사용자 정의

Stage의 표시 객체에 대해 마우스 커서(마우스 포인터)를 숨기거나 교체할 수 있습니다. 마우스 커서를 숨기려면 `Mouse.hide()` 메서드를 호출합니다. `Mouse.hide()`를 호출하고 해당 스테이지에서 `MouseEvent.MOUSE_MOVE` 이벤트를 수신하는 한편 표시 객체(사용자 정의 커서)의 좌표를 해당 이벤트의 `stageX` 및 `stageY` 속성으로 설정함으로써 커서를 사용자 정의할 수 있습니다. 다음 예제에서는 이 작업을 실행하는 방법을 보여 줍니다.

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
  cursor.x = event.stageX;
  cursor.y = event.stageY;
}
```

컨텍스트 메뉴 사용자 정의

InteractiveObject 클래스로부터 상속되는 모든 객체에는 사용자가 SWF 파일에서 마우스 오른쪽 버튼을 클릭할 경우 표시되는 고유한 컨텍스트 메뉴가 포함될 수 있습니다. [앞으로 이동], [뒤로 이동], [인쇄], [품질], [확대/축소] 등의 몇 가지 명령은 기본적으로 포함됩니다.

[설정] 및 [정보] 명령을 제외한 모든 기본 명령은 메뉴에서 제거할 수 있습니다. 스테이지 속성 `showDefaultContextMenu`를 `false`로 설정하면 컨텍스트 메뉴에서 이러한 명령이 제거됩니다.

특정한 표시 객체의 컨텍스트 메뉴를 사용자 정의하려면 ContextMenu 클래스의 새 인스턴스를 만들고 `hideBuiltInItems()` 메서드를 호출한 다음 이 인스턴스를 해당하는 DisplayObject 인스턴스의 `contextMenu` 속성에 지정합니다. 다음 예제에서는 동적으로 그려지는 정사각형에 임의의 색상으로 변경할 수 있는 컨텍스트 메뉴 명령을 제공합니다.

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
```

```
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
  square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
  return new ColorTransform(Math.random(), Math.random(),
    Math.random(),1,(Math.random() * 512) - 255,
    (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

포커스 관리

대화형 객체는 프로그래밍 또는 사용자 액션을 통해 포커스를 받을 수 있습니다. 어떤 방법을 택하든, 포커스를 설정하면 객체의 `focus` 속성이 `true`로 바뀝니다. 또한 `tabEnabled` 속성이 `true`로 설정되어 있는 경우에는 Tab 키를 눌러 특정 객체의 포커스를 다른 객체로 넘길 수 있습니다. 다음과 같은 경우를 제외하고, `tabEnabled`의 기본값은 `false`입니다.

- SimpleButton 객체의 경우 이 값이 `true`입니다.
- 입력 텍스트 필드의 경우 이 값이 `true`입니다.
- `buttonMode`가 `true`로 설정된 Sprite 객체 또는 MovieClip 객체의 경우 이 값이 `true`입니다.

이러한 경우 `FocusEvent.FOCUS_IN` 또는 `FocusEvent.FOCUS_OUT`에 대한 리스너를 추가하여 포커스 변경 시의 비헤이비어를 추가할 수 있습니다. 이는 특히 텍스트 필드 및 양식에 유용하지만 스프라이트, 무비 클립 또는 InteractiveObject 클래스로부터 상속되는 모든 객체에서도 사용할 수 있습니다. 다음 예제에는 Tab 키를 사용하여 포커스를 순환할 수 있도록 설정하는 방법과 이후의 포커스 이벤트에 응답하는 방법이 나와 있습니다. 이 경우 각 정사각형에 포커스가 설정되면 색상이 변경됩니다.

예제

Flash 제작 도구는 키보드 단축키를 사용하여 포커스를 관리합니다. 따라서 포커스 관리를 제대로 시뮬레이션하려면 Flash가 아닌 브라우저에서 SWF 파일을 테스트해야 합니다.

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
```

```
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
  for (j = 0; j < cols; j++)
  {
    createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
  }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
  var square:Sprite = new Sprite();
  square.graphics.beginFill(0x000000);
  square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
  square.graphics.endFill();
  square.x = startX;
  square.y = startY;
  square.tabEnabled = true;
  square.tabIndex = tabNumber;
  square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
  addChild(square);
}
function changeColor(event:FocusEvent):void
{
  e.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
  // 빨강, 녹색 및 파랑 색상 채널에 대한 무작위 값을 생성합니다.
  var red:Number = (Math.random() * 512) - 255;
  var green:Number = (Math.random() * 512) - 255;
  var blue:Number = (Math.random() * 512) - 255;

  // 무작위 색상으로 ColorTransform 객체를 만들어 반환합니다.
  return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

예제: WordSearch

이 예제는 마우스 이벤트 처리를 통한 사용자 상호 작용에 대한 것입니다. 사용자는 바둑판 무늬에 배열된 임의의 문자로 가능한 한 많은 단어를 조합해야 하며, 문자를 가로 또는 세로로 이동하여 철자를 맞추고 같은 문자를 두 번 사용할 수 없습니다. 이 예제에서는 다음과 같은 ActionScript 3.0의 기능이 사용됩니다.

- 구성 요소 배열을 동적으로 구성
- 마우스 이벤트에 응답
- 사용자 상호 작용에 기초하여 점수 유지 관리

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. WordSearch 응용 프로그램 파일은 Samples/WordSearch 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
WordSearch.as	응용 프로그램의 기본 기능을 제공하는 클래스입니다.
WordSearch.fla	Flash용 기본 응용 프로그램 파일입니다.
dictionary.txt	입력한 단어가 득점 가능하며 정확하게 입력되었는지 확인하는 데 사용되는 파일입니다.

사전 로드

단어 찾기와 관련된 게임을 만들기 위해서는 사전이 필요합니다. 이 예제에는 캐리지 리턴으로 구분된 단어 목록을 포함하는 dictionary.txt 텍스트 파일이 포함되어 있습니다. `words`라는 배열을 만들면 `loadDictionary()` 함수에서 이 파일을 요청하며, 로드된 파일은 긴 문자열이 됩니다. 그러면 `split()` 메서드를 사용하여 이 문자열을 단어 배열로 파싱할 수 있습니다. 이때 캐리지 리턴(문자 코드 10)의 각 인스턴스에서 줄바꿈됩니다. 이 파싱 작업은 `dictionaryLoaded()` 함수에서 실행됩니다.

```
words = dictionaryText.split(String.fromCharCode(10));
```

사용자 인터페이스 만들기

단어가 저장되면 사용자 인터페이스를 설정할 수 있습니다. 이때 Button 인스턴스를 두 개 만들어야 하는데, 하나는 단어를 제출할 때 사용되고 다른 하나는 현재 철자를 맞추고 있는 단어를 지울 때 사용됩니다. 각 상황에서 해당 버튼이 브로드캐스팅하는 `MouseEvent.CLICK` 이벤트를 수신한 다음 함수를 호출하여 사용자 입력에 응답해야 합니다. `setupUI()` 함수에서 이 코드는 두 개의 버튼에 리스너를 만듭니다.

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

게임 보드 생성

게임 보드는 바둑판 무늬로 된 임의의 문자 배열입니다. `generateBoard()` 함수에서 한 루프 내에 다른 루프를 중첩시켜 2차원 배열을 만듭니다. 첫 번째 루프에서는 행이 증가되고 두 번째 루프에서는 한 행당 총 열 수가 증가됩니다. 이러한 행과 열에 의해 만들어진 각 셀에는 보드의 문자를 나타내는 버튼이 포함됩니다.

```
private function generateBoard(startX:Number,
  startY:Number,
  totalRows:Number,
  totalCols:Number,
  buttonSize:Number):void
{
  buttons = new Array();
  var colCounter:uint;
  var rowCounter:uint;
  for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
  {
    for (colCounter = 0; colCounter < totalCols; colCounter++)
    {
      var b:Button = new Button();
      b.x = startX + (colCounter*buttonSize);
      b.y = startY + (rowCounter*buttonSize);
      b.addEventListener(MouseEvent.CLICK, letterClicked);
      b.label = getRandomLetter().toUpperCase();
      b.setSize(buttonSize,buttonSize);
      b.name = "buttonRow"+rowCounter+"Col"+colCounter;
      addChild(b);

      buttons.push(b);
    }
  }
}
```

오직 한 행의 `MouseEvent.CLICK` 이벤트에만 리스너를 추가해도 이것이 `for` 루프이므로 각각의 Button 인스턴스에 지정됩니다. 또한 각 버튼에는 해당 행 및 열 위치에서 파생된 이름이 지정되므로 나중에 코드에서 각 버튼의 행과 열을 참조하기가 쉽습니다.

사용자 입력을 통해 단어 작성

가로 또는 세로로 인접한 문자를 선택하여 철자를 맞출 수 있지만 같은 문자를 두 번 사용할 수 없습니다. 한 번 클릭할 때마다 마우스 이벤트가 생성되는데 이 시점에서 사용자가 입력하는 단어를 점검하여 이전에 클릭한 문자에서 적절하게 이어지는지 확인해야 합니다. 제대로 연결되지 않으면 이전 단어는 제거되고 새 단어가 시작됩니다. 이 검사 작업은 `isLegalContinuation()` 메서드에서 실행됩니다.

```
private function isLegalContinuation(prevButton:Button,
  currButton:Button):Boolean
```

```
{
  var currButtonRow:Number =
  Number(currButton.name.charAt(currButton.name. indexOf("Row") + 3));
  var currButtonCol:Number =
  Number(currButton.name.charAt(currButton.name.indexOf("Col") + 3));
  var prevButtonRow:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Row") + 3));
  var prevButtonCol:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Col") + 3));

  return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow -
  currButtonRow) <= 1) ||
       (prevButtonRow == currButtonRow && Math.abs(prevButtonCol -
  currButtonCol) <= 1));
}
```

String 클래스의 `charAt()` 및 `indexOf()` 메서드는 현재 클릭한 버튼과 이전에 클릭한 버튼 둘 다에서 적절한 행과 열을 얻습니다. 이때 `isLegalContinuation()` 메서드는 행 또는 열이 변경되지 않았거나 변경된 행 또는 열의 위치가 이전 행 또는 열에서 한 칸 이내일 경우 `true`를 반환합니다. 대각선으로 단어의 철자를 맞출 수 있도록 게임 규칙을 변경하려면 변경되지 않은 행 또는 열에 대해 검사 작업을 제거해야 하며, 이 경우 마지막 행이 다음과 같습니다.

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) &&
  Math.abs(prevButtonCol - currButtonCol) <= 1));
```

제안 단어 검사

게임에 대한 코드를 완성하려면 제안 단어 검사 및 점수 기록을 위한 메커니즘이 필요합니다. `searchForWord()` 메서드에는 두 가지 메커니즘이 모두 들어 있습니다.

```
private function searchForWord(str:String):Number
{
  if (words && str)
  {
    var i:uint = 0
    for (i = 0; i < words.length; i++)
    {
      var thisWord:String = words[i];
      if (str == words[i])
      {
        return i;
      }
    }
    return -1;
  }
  else
  {
    trace("WARNING: cannot find words, or string supplied is null");
```

```
    }
    return -1;
}
```

이 함수는 사전에서 모든 단어를 반복 탐색합니다. 사용자의 단어가 사전의 단어와 일치하면 사전에서의 위치가 반환됩니다. 그런 다음 `submitWord()` 메서드가 응답을 검사하여 위치가 유효한 경우 점수를 업데이트합니다.

사용자 정의

클래스 시작 부분에는 몇 개의 상수가 있습니다. 이러한 변수를 수정하여 게임을 수정할 수 있습니다. 예를 들어 `TOTAL_TIME` 변수의 값을 늘려 게임 플레이 시간을 변경할 수 있습니다. 또한 `PERCENT_VOWELS` 변수의 값도 약간 늘릴 수 있는데, 이렇게 하면 단어를 찾을 가능성이 커집니다.

제 22 장

네트워킹 및 통신

22

이 장에서는 SWF 파일을 사용하여 외부 파일 및 기타 Adobe Flash Player 9 인스턴스와 통신하는 방법에 대해 설명합니다. 또한 외부 소스에서 데이터를 로드하고, Java 서버와 Flash Player 간에 메시지를 전송하고, FileReference 및 FileReferenceList 클래스를 사용하여 파일을 업로드 및 다운로드하는 방법에 대해 설명합니다.

목차

네트워킹 및 통신의 기초

네트워킹 및 통신 소개

복잡한 ActionScript 응용 프로그램을 작성할 때 서버측 스크립트와 통신하거나 외부 XML 또는 텍스트 파일에서 데이터를 로드해야 하는 경우가 자주 있습니다. flash.net 패키지에는 원격 URL에서 내용을 로드하고 다른 Flash Player 인스턴스와 통신하며 원격 웹 사이트에 연결하는 등 인터넷을 통해 데이터를 송수신할 수 있는 클래스가 포함되어 있습니다.

ActionScript 3.0에서는 URLLoader 및 URLRequest 클래스를 사용하여 외부 파일을 로드할 수 있습니다. 그런 다음 로드된 데이터 유형에 따라 특정 클래스를 사용하여 데이터에 액세스할 수 있습니다. 예를 들어, 원격 내용이 이름-값 쌍 형식으로 지정된 경우 URLVariables 클래스를 사용하여 서버 결과를 구문 분석합니다. 그렇지 않고 URLLoader 및 URLRequest 클래스를 사용하여 로드된 파일이 원격 XML 문서인 경우, XML 클래스의 생성자, XMLDocument 클래스의 생성자 또는 `XMLDocument.parseXML()` 메서드를 사용하여 XML 문서를 구문 분석할 수 있습니다. 그러면 URLVariables, XML 또는 원격 데이터의 구문 분석 및 작업에 사용하는 다른 클래스 중 어떤 것을 사용하는지에 관계없이 외부 파일을 로드하는 코드가 동일하므로 ActionScript 코드를 단순화할 수 있습니다.

flash.net 패키지에는 다른 유형의 원격 통신을 위한 클래스도 들어 있습니다. 여기에는 서버에서 파일을 업로드하고 다운로드할 수 있는 FileReference 클래스, 소켓 연결을 통해 원격 컴퓨터와 직접 통신할 수 있는 Socket 및 XMLSocket 클래스, Flash 전용 서버 리소스(예: Flash Media Server 및 Flash Remoting Server)와 통신하고 비디오 파일을 로드하는 데 사용되는 NetConnection 및 NetStream 클래스 등이 포함됩니다.

마지막으로 flash.net 패키지에는 사용자의 로컬 컴퓨터에서 통신하는 데 필요한 클래스가 들어 있습니다. 한 대의 컴퓨터에서 실행되는 둘 이상의 SWF 파일 간에 통신을 가능하게 하는 LocalConnection 클래스와, 사용자의 컴퓨터에 데이터를 저장하고 나중에 해당 데이터가 응용 프로그램에 반환될 때 검색할 수 있도록 하는 SharedObject 클래스 등이 여기에 포함됩니다.

일반적인 네트워킹 및 통신 작업

다음은 ActionScript에서의 외부 통신과 관련하여 실행할 수 있는 가장 일반적인 작업으로, 이 장에서 설명하는 내용입니다.

- 외부 파일 또는 서버 스크립트에서 데이터 로드
- 서버 스크립트에 데이터 보내기
- 다른 로컬 SWF 파일과 통신
- 이진 소켓 연결을 사용한 작업
- XML 소켓으로 통신

- 영구 로컬 데이터 저장
- 서버에 파일 업로드
- 서버에서 사용자의 컴퓨터로 파일 다운로드

중요한 개념 및 용어

다음 참조 목록에는 이 장에 사용된 중요한 용어가 포함되어 있습니다.

- 외부 데이터: SWF 파일 외부의 일부 양식에 저장되는 데이터로 필요할 때 SWF 파일로 로드됩니다. 이 데이터는 직접 로드한 파일에 저장하거나, 서버에서 실행되는 스크립트나 프로그램을 호출하여 가져온 데이터베이스나 다른 양식에 저장할 수 있습니다.
- URL 인코딩된 변수: URL 인코딩 형식을 사용하면 여러 가지 변수(변수 이름과 값 쌍)를 하나의 텍스트 문자열로 나타낼 수 있습니다. 개별 변수는 `name=value` 형식으로 작성됩니다. 각 변수(이름-값 쌍)는 앰퍼샌드 문자로 구분됩니다(예: `variable1=value1&variable2=value2`). 이러한 방법으로 무제한의 변수를 하나의 메시지로 보낼 수 있습니다.
- MIME 형식: 인터넷 통신에서 제공된 파일 유형을 확인하는 데 사용되는 표준 코드입니다. 모든 파일 유형에는 해당 유형을 식별하는 데 사용되는 특정 코드가 있습니다. 컴퓨터(예: 웹 서버 또는 사용자의 Flash Player 인스턴스)에서 파일 또는 메시지를 보낼 때 보내는 파일 유형을 지정합니다.
- HTTP(Hypertext Transfer Protocol): 인터넷을 통해 전송되는 웹 페이지 및 다양한 유형의 기타 내용을 전달하기 위한 표준 포맷입니다.
- 요청 메서드: Flash Player 또는 웹 브라우저 같은 프로그램에서 메시지(HTTP 요청이라고 함)를 웹 서버에 보낼 때 전송되는 모든 데이터는 GET과 POST의 두 가지 *요청 메서드* 중 한 가지 방식을 사용하여 요청에 포함할 수 있습니다. 서버 끝에서 요청을 받는 프로그램은 데이터를 찾기 위해 요청의 적절한 부분을 조회해야 하므로, ActionScript에서 데이터를 보내는 데 사용되는 요청 메서드는 서버에서 해당 데이터를 읽을 때 사용되는 요청 메서드와 일치해야 합니다.
- 소켓 연결: 두 컴퓨터 간의 통신을 위한 영구 연결입니다.
- 업로드: 파일을 다른 컴퓨터로 보냅니다.
- 다운로드: 다른 컴퓨터에서 파일을 가져옵니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플 중 일부는 외부 데이터를 로드하거나 다른 유형의 통신을 수행합니다. 이러한 예제에 `trace()` 함수 호출이 포함되어 예제 실행 결과를 [출력] 패널에 표시하는 경우도 있습니다. 다른 예제는 실제로 몇 가지 기능(예: 서버에 파일 업로드)을 수행합니다. 이러한 예제를 테스트하려면 SWF와 상호 작용하고 해당 예제가 요청된 작업을 수행하는지 확인해야 합니다.

코드 예제는 두 가지 범주로 구분됩니다. 예제 샘플 중 일부는 코드가 독립 실행형 스크립트인 경우(예: Flash 문서의 키프레임에 첨부된 상태)를 가정하여 작성되었습니다. 이러한 예제를 테스트하려면:

1. 새 Flash 문서를 만듭니다.
2. 타임라인의 프레임 1에서 키프레임을 선택하고 [액션] 패널을 엽니다.
3. [스크립트] 창에 코드 샘플을 복사합니다.
4. 주 메뉴에서 [컨트롤] > [무비 테스트]를 선택하여 SWF 파일을 만들고 예제를 테스트합니다.

다른 예제 코드 샘플은 클래스로 작성되었으며 예제 클래스가 Flash 문서에 대한 문서 클래스의 역할을 수행합니다. 이러한 예제를 테스트하려면:

1. 빈 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 새 ActionScript 파일을 만들고 Flash 문서와 같은 디렉토리에 저장합니다. 파일 이름은 코드 샘에 있는 클래스의 이름과 일치해야 합니다. 예를 들어, 코드 샘플에 "UploadTest"라는 클래스가 정의된 경우 ActionScript 파일을 "UploadTest.as"로 저장합니다.
3. ActionScript 파일에 코드 샘플을 복사하고 파일을 저장합니다.
4. Flash 문서에서 스테이지 또는 페이스트보드의 빈 부분을 클릭하여 문서 속성 관리자를 활성화합니다.
5. 텍스트에서 복사한 ActionScript 클래스의 이름을 속성 관리자의 [문서 클래스] 필드에 입력합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행하고 예제를 테스트합니다.

마지막으로 이 장의 일부 예제는 서버에서 실행되는 프로그램과 상호 작용합니다. 이러한 예제에는 예제 테스트에 필요한 서버 프로그램을 만드는 데 사용할 수 있는 코드가 포함되어 있습니다. 이러한 예제를 테스트하려면 웹 서버 컴퓨터에서 해당 응용 프로그램을 설정해야 합니다.

외부 데이터를 사용한 작업

ActionScript 3.0에는 외부 소스에서 데이터를 로드하기 위한 메커니즘이 있습니다. 외부 소스는 텍스트 파일과 같은 정적 내용일 수도 있고 데이터베이스에서 데이터를 가져오는 웹 스크립트와 같은 동적 내용일 수 있습니다. 데이터의 형식은 다양한 방법으로 지정할 수 있으며 ActionScript는 데이터의 암호를 해독하고 데이터에 액세스하는 기능을 제공합니다. 데이터를 가져오는 프로세스의 일환으로 데이터를 외부 서버에 보낼 수도 있습니다.

URLLoader 및 URLVariables 클래스 사용

ActionScript 3.0에서는 URLLoader 및 URLVariables 클래스를 사용하여 외부 데이터를 로드합니다. URLLoader 클래스는 URL에서 텍스트, 이진 데이터 또는 URL 인코딩된 변수 형식으로 데이터를 다운로드합니다. URLLoader 클래스는 텍스트 파일, XML 또는 동적 데이터 기반 ActionScript 응용 프로그램에서 사용되는 기타 정보를 다운로드할 때 유용합니다.

URLLoader 클래스는 ActionScript 3.0 고급 이벤트 처리 모델을 사용하므로 `complete`, `httpStatus`, `ioError`, `open`, `progress` 및 `securityError` 등의 이벤트를 수신할 수 있습니다. 새로운 이벤트 처리 모델은 ActionScript 2.0의 `LoadVars.onData`, `LoadVars.onHTTPStatus` 및 `LoadVars.onLoad` 이벤트 핸들러 지원 기능보다 크게 향상되어 오류 및 이벤트를 더욱 효율적으로 처리할 수 있습니다. 이벤트 처리에 대한 자세한 내용은 제10장, “이벤트 처리”를 참조하십시오.

이전 버전 ActionScript의 XML 및 LoadVars 클래스와 마찬가지로 URLLoader URL의 데이터는 다운로드가 완료될 때까지 사용할 수 없습니다. 전달되는 `flash.events.ProgressEvent.PROGRESS` 이벤트를 수신하여 다운로드 진행률(로드된 바이트 수 및 전체 바이트 수)을 모니터링할 수 있습니다. 단, 파일이 너무 빨리 로드되면 `ProgressEvent.PROGRESS` 이벤트가 전달되지 않을 수 있습니다. 파일을 다운로드하면 `flash.events.Event.COMPLETE` 이벤트가 전달됩니다. 로드된 데이터는 UTF-8 또는 UTF-16 인코딩에서 문자열로 디코딩됩니다.

예외	`URLRequest.contentType`**에 아무 값도 설정되어 있지 않으면 값이** `application/x-www-form-urlencoded`**로 전송됩니다.**

`URLLoader.load()` 메서드 및 URLLoader 클래스 생성자(선택 항목)는 URLRequest 인스턴스인 `request` 하나만 매개 변수로 사용합니다. URLRequest 인스턴스에는 대상 URL, 요청 메서드(`GET` 또는 `POST`), 추가 헤더 정보 및 MIME 유형(예: XML 내용을 업로드할 때)과 같은 단일 HTTP 요청의 모든 정보가 들어 있습니다.

예를 들어, XML 패킷을 서버측 스크립트에 업로드할 때 다음과 같은 ActionScript 3.0 코드를 사용할 수 있습니다.

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
  <login>
```

```
    <time>{secondsUTC}</time>
    <username>Ernie</username>
    <password>guru</password>
  </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
  loader.load(request);
}
catch (error:ArgumentError)
{
  trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
  trace("A SecurityError has occurred.");
}
```

앞의 코드 예제에서는 `dataXML`이라는 XML 인스턴스(서버에 보낼 XML 패킷 포함)를 만듭니다. 다음으로 URLRequest `contentType` 속성을 `"text/xml"`로 설정하고 URLRequest `data` 속성을 XML 패킷의 내용으로 설정합니다. 이 내용은 `XML.toXMLString()` 메서드를 사용하여 문자열로 변환됩니다. 마지막으로 `URLLoader.load()` 메서드를 사용하여 새로운 URLLoader 인스턴스를 만들고 원격 스크립트에 요청을 보냅니다.

URL 요청에서 전달할 매개 변수를 지정할 수 있는 방법은 다음 세 가지입니다.

- URLVariables 생성자 내에서 지정
- `URLVariables.decode()` 메서드 내에서 지정
- URLVariables 객체 자체에 포함된 특정 속성에 따라 지정

URLVariables 생성자나 `URLVariables.decode()` 메서드 내에서 변수를 정의할 때 앰퍼샌드 문자는 특별한 의미가 있는 구분 기호로 사용되므로 URL 인코딩해야 합니다. 예를 들어, 앰퍼샌드(`&`)는 매개 변수 구분 기호로 사용되므로 전달할 때 `%26`으로 변경하여 URL 인코딩해야 합니다.

외부 문서에서 데이터 로드

ActionScript 3.0으로 동적 응용 프로그램을 작성할 때 외부 파일이나 서버측 스크립트에서 데이터를 로드하는 것이 좋습니다. 그러면 ActionScript 파일을 편집하거나 다시 컴파일하지 않고도 동적 응용 프로그램을 작성할 수 있습니다. 예를 들어, "오늘의" 팁을 표시하는 응용 프로그램을 작성하는 경우 하루에 한 번씩 데이터베이스에서 임의로 하나의 팁을 가져와 텍스트 파일로 저장하는 서버측 스크립트를 작성할 수 있습니다. 그러면 ActionScript 응용 프로그램에서 매번 데이터베이스를 쿼리하지 않고 정적 텍스트 파일의 내용을 로드할 수 있습니다.

다음 코드 예제에서는 외부 텍스트 파일인 params.txt의 내용을 로드하는 URLRequest 및 URLLoader 객체를 만듭니다.

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

앞의 코드 예제를 다음과 같이 간단하게 작성할 수 있습니다.

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

기본적으로 요청 메서드를 정의하지 않으면 Flash Player에서 HTTP `GET` 메서드를 사용하여 내용을 로드합니다. `POST` 메서드를 사용하여 데이터를 전송하려면 정적 상수 `URLRequestMethod.POST`를 사용하여 다음 코드와 같이 `request.method` 속성을 `POST`로 설정해야 합니다.

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

런타임에 로드되는 외부 문서 params.txt에는 다음 데이터가 포함됩니다.

```
monthNames=January,February,March,April,May,June,July,August,September,Oct
  ober,November,December&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday
  ,Friday,Saturday
```

`monthNames`와 `dayNames`라는 두 개의 매개 변수가 포함됩니다. 각 매개 변수에는 쉼표로 구분되어 문자열로 구문 분석된 목록이 들어 있습니다. `String.split()` 메서드를 사용하여 이 목록을 배열로 나눌 수 있습니다.

팁	코드를 읽거나 디버깅하기가 어려워지므로 외부 데이터 파일에서 예약어나 언어 구문을 변수 이름으로 사용하지 마십시오.

데이터가 로드되면 Event.COMPLETE 이벤트가 전달되고 다음 코드에서 보듯이 URLLoader의 data 속성에서 외부 문서의 내용을 사용할 수 있게 됩니다.

```
private function completeHandler(event:Event):void
{
  var loader2:URLLoader = URLLoader(event.target);
  trace(loader2.data);
}
```

원격 문서에 이름-값 쌍이 있으면 다음과 같이 로드된 파일의 내용에서 전달하는 방식으로 URLVariables 클래스를 사용하여 데이터의 구문을 분석할 수 있습니다.

```
private function completeHandler(event:Event):void
{
  var loader2:URLLoader = URLLoader(event.target);
  var variables:URLVariables = new URLVariables(loader2.data);
  trace(variables.dayNames);
}
```

외부 파일의 각 이름-값 쌍은 URLVariables 객체에서 속성으로 만들어집니다. 앞의 코드 샘플에서 변수 객체의 각 속성은 문자열로 처리됩니다. 이름-값 쌍의 값이 항목의 목록이면 다음과 같이 String.split() 메서드를 호출하여 문자열을 배열로 변환할 수 있습니다.

```
var dayNameArray:Array = variables.dayNames.split(",");
```

팁 **외부 텍스트 파일에서 숫자 데이터를 로드하는 경우 int(), uint() 또는 Number() 등 최상위 함수를 사용하여 값을 숫자 값으로 변환해야 합니다.**

원격 파일의 내용을 문자열로 로드하여 새 URLVariables 객체를 만드는 대신 URLLoader.dataFormat 속성을 URLLoaderDataFormat 클래스에 있는 정적 속성 중 하나로 설정할 수 있습니다. URLLoader.dataFormat 속성에 사용할 수 있는 값은 다음 세 가지입니다.

- URLLoaderDataFormat.BINARY - ByteArray 객체에 저장된 이진 데이터가 URLLoader.data 속성에 포함됩니다.
- URLLoaderDataFormat.TEXT - 문자열 객체의 텍스트가 URLLoader.data 속성에 포함됩니다.
- URLLoaderDataFormat.VARIABLES - URLVariables 객체에 저장된 URL 인코딩된 변수가 URLLoader.data 속성에 포함됩니다.

다음 코드는 URLLoader.dataFormat 속성을 URLLoaderDataFormat.VARIABLES로 설정하는 경우 로드된 데이터가 자동으로 URLVariables 객체로 구문 분석되는 방법을 보여 줍니다.

```
package
{
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;

  public class URLLoaderDataFormatExample extends Sprite
  {
    public function URLLoaderDataFormatExample()
    {
      var request:URLRequest = new URLRequest("http://
  www.[yourdomain].com/params.txt");
      var variables:URLLoader = new URLLoader();
      variables.dataFormat = URLLoaderDataFormat.VARIABLES;
      variables.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        variables.load(request);
      }
      catch (error:Error)
      {
        trace("Unable to load URL: " + error);
      }
    }
    private function completeHandler(event:Event):void
    {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
    }
  }
}
```

예외 | URLLoader.dataFormat의 기본값은 URLLoaderDataFormat.TEXT입니다.

다음 예제에서 보듯이 외부 파일에서 XML을 로드하는 것은 URLVariables를 로드하는 것과 같습니다. URLRequest 인스턴스와 URLLoader 인스턴스를 만들고 이러한 인스턴스를 사용하여 원격 XML 문서를 다운로드할 수 있습니다. 파일이 완전히 다운로드되면 Event.COMPLETE 이벤트가 전달되고 외부 파일의 내용이 XML 인스턴스로 변환됩니다. 이 인스턴스는 XML 메서드와 속성을 사용하여 구문 분석할 수 있습니다.

```
package
{
  import flash.display.Sprite;
  import flash.errors.*;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLRequest;

  public class ExternalDocs extends Sprite
  {
    public function ExternalDocs()
    {
      var request:URLRequest = new URLRequest("http://
  www.[yourdomain].com/data.xml");
      var loader:URLLoader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        loader.load(request);
      }
      catch (error:ArgumentError)
      {
        trace("An ArgumentError has occurred.");
      }
      catch (error:SecurityError)
      {
        trace("A SecurityError has occurred.");
      }
    }
    private function completeHandler(event:Event):void
    {
      var dataXML:XML = XML(event.target.data);
      trace(dataXML.toXMLString());
    }
  }
}
```

외부 스크립트와 통신

외부 데이터 파일 로드 외에 URLVariables 클래스를 사용하여 변수를 서버측 스크립트로 전송하고 서버의 응답을 처리할 수도 있습니다. 예를 들어, 게임을 프로그래밍하는 경우 사용자의 점수를 서버로 전송하여 고득점 목록에 추가해야 할지 여부를 계산하거나 사용자 로그인 정보를 서버로 전송하여 확인하려는 경우에 유용합니다. 서버측 스크립트는 사용자 이름과 암호를 처리하고, 데이터베이스에서 해당 이름과 암호가 맞는지 확인하며, 사용자가 제공한 자격 증명이 유효한지 여부를 확인하여 반환할 수 있습니다.

다음 코드 예제에서는 `variables`라는 URLVariables 객체를 만듭니다. 이 객체는 `name`이라는 새 변수를 만듭니다. 그 다음에는 변수를 보낼 서버측 스크립트의 URL을 지정하는 URLRequest 객체를 만듭니다. 그런 다음 URLRequest 객체의 `method` 속성을 설정하여 변수를 HTTP `POST` 요청으로 보냅니다. URLVariables 객체를 URL 요청에 추가하려면 URLRequest 객체의 `data` 속성을 이전에 만든 URLVariables 객체로 설정합니다. 마지막으로 URLLoader 인스턴스가 만들고 `URLLoader.load()` 메서드를 호출하여 요청을 시작합니다.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
  loader.load(request);
}
catch (error:Error)
{
  trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
  trace(event.target.data.welcomeMessage);
}
```

다음 코드에는 앞의 예제에서 사용된 Adobe ColdFusion® greeting.cfm 문서의 내용이 포함되어 있습니다.

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
  <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

다른 Flash Player 인스턴스에 연결

LocalConnection 클래스를 사용하면 HTML 컨테이너의 SWF나 포함된 플레이어 또는 독립 실행형 플레이어의 SWF 등 서로 다른 Flash Player 인스턴스 간에 통신할 수 있습니다. 따라서 웹 브라우저에서 실행되는 SWF 파일이나 데스크톱 응용 프로그램에 포함된 SWF 파일 등 Flash Player 인스턴스 간에 데이터를 공유할 수 있는 매우 다양한 응용 프로그램을 만들 수 있습니다.

LocalConnection 클래스

LocalConnection 클래스를 사용하면 `fscommand()` 메서드나 JavaScript를 사용하지 않고 다른 SWF 파일에 명령을 보낼 수 있는 SWF 파일을 개발할 수 있습니다. LocalConnection 객체는 동일한 클라이언트 시스템에서 실행되는 SWF 파일 간에만 통신할 수 있습니다. 하지만 서로 다른 응용 프로그램에서 실행되는 경우는 가능합니다. 예를 들어, 프로젝터에서 로컬 정보를 유지하고 원격으로 브라우저 기반 SWF에 연결한 상태로 파일 브라우저에서 실행되는 SWF 파일과 프로젝터에서 실행되는 SWF 파일이 정보를 공유할 수 있습니다. 프로젝터는 독립 실행형 응용 프로그램으로 실행할 수 있는 형식으로 저장된 SWF 파일로, 실행 파일 내에 Flash Player가 포함되어 있으므로 Flash Player를 설치할 필요가 없습니다.

LocalConnection 객체는 다음과 같이 여러 ActionScript 버전을 사용하여 SWF 간에 통신하는 데 사용할 수 있습니다.

- ActionScript 3.0 LocalConnection 객체는 ActionScript 1.0 또는 2.0으로 만들어진 LocalConnection 객체와 통신할 수 있습니다.
- ActionScript 1.0 또는 2.0 LocalConnection 객체는 ActionScript 3.0으로 만들어진 LocalConnection 객체와 통신할 수 있습니다.

Flash Player는 다른 버전의 LocalConnection 객체 간의 통신을 자동으로 처리합니다.

LocalConnection 객체를 가장 간단하게 사용할 수 있는 방법은 동일한 도메인에 위치한 LocalConnection 객체 사이에서만 통신을 허용하는 것입니다. 그러면 보안 문제에 대해 걱정하지 않아도 됩니다. 서로 다른 도메인 사이에서 통신해야 하는 경우, 보안 문제를 해결할 수 있는 몇 가지 방법이 있습니다. 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 LocalConnection 클래스 샘플의 `allowDomain()` 및 `domain` 항목과 `send()` 메서드의 `connectionName` 매개 변수에 대한 설명을 참조하십시오.

참고 LocalConnection 객체를 사용하여 하나의 SWF 파일에 포함된 데이터를 송수신할 수 있으나 좋은 방법은 아닙니다. 대신 공유 객체를 사용해야 합니다.

LocalConnection 객체에 콜백 메서드를 추가하는 방법은 다음 세 가지입니다.

- LocalConnection 클래스를 하위 클래스로 사용하고 메서드 추가
- LocalConnection.client 속성을 이 메서드를 구현하는 객체로 설정
- LocalConnection을 확장하는 동적 클래스를 만들고 동적으로 메서드 추가

콜백 메서드를 추가하는 첫 번째 방법은 LocalConnection 클래스를 확장하는 것입니다. 동적으로 사용자 정의 클래스를 LocalConnection 인스턴스에 추가하지 않고 사용자 정의 클래스 내에 메서드를 정의합니다. 이 방법은 다음 코드에서 확인할 수 있습니다.

```
package
{
  import flash.net.LocalConnection;
  public class CustomLocalConnection extends LocalConnection
  {
    public function CustomLocalConnection(connectionName:String)
    {
      try
      {
        connect(connectionName);
      }
      catch (error:ArgumentError)
      {
        // 서버가 이미 만들어지고 연결되었습니다 .
      }
    }
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

DynamicLocalConnection 클래스의 새 인스턴스를 만들려면 다음 코드를 사용할 수 있습니다.

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

콜백 메서드를 추가하는 두 번째 방법은 LocalConnection.client 속성을 사용하는 것입니다. 다음 코드와 같이 사용자 정의 클래스를 만들고 새 인스턴스를 client 속성에 할당하는 것입니다.

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

LocalConnection.client 속성은 호출하면 안 되는 객체 콜백 메서드를 나타냅니다. 앞의 코드에서 client 속성은 사용자 정의 클래스 CustomClient의 새 인스턴스로 설정되었습니다. client 속성의 기본값은 현재 LocalConnection 인스턴스입니다. 창 하나의 버튼이 두 번째 창의 보기를 전환하는 응용 프로그램에서 메서드 설정은 같지만 다르게 작동하는 두 개의 데이터 핸들러가 있는 경우 client 속성을 사용할 수 있습니다.

CustomClient 클래스를 만들려면 다음 코드를 사용할 수 있습니다.

```
package
{
  public class CustomClient extends Object
  {
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

동적 클래스를 만들고 동적으로 메서드를 추가하는 등 콜백 메서드를 추가하는 세 번째 방법은 다음 코드에서 보듯이 이전 버전의 ActionScript에서 LocalConnection 클래스를 사용하는 것과 매우 비슷합니다.

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

다음 코드를 사용하여 이 클래스에 콜백 메서드를 동적으로 추가할 수 있습니다.

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// 여기에 코드를 추가하십시오 .
public function onMethod(timeString:String):void
{
  trace("onMethod called at: " + timeString);
}
```

앞에서 설명한 콜백 메서드 추가 방법은 코드를 옮기기가 매우 어려우므로 사용하지 않는 것이 좋습니다. 뿐만 아니라 동적 속성에 액세스하는 경우 봉인된 속성에 액세스하는 경우보다 속도가 크게 느리므로 이 방법을 사용하여 로컬 연결을 만들면 성능 문제가 발생할 수 있습니다.

두 Flash Player 인스턴스 간 메시지 보내기

LocalConnection 클래스를 사용하여 여러 가지 Flash Player의 인스턴스 간에 통신합니다. 예를 들어, 웹 페이지에 여러 Flash Player 인스턴스가 있거나 Flash Player 인스턴스가 팝업 창의 Flash Player 인스턴스에서 데이터를 가져오도록 할 수 있습니다.

다음 코드에서는 서버처럼 작동하고 다른 Flash Player 인스턴스에서 수신 호출을 허용하는 로컬 연결 객체를 정의합니다.

```
package
{
  import flash.net.LocalConnection;
  import flash.display.Sprite;
  public class ServerLC extends Sprite
  {
    public function ServerLC()
    {
      var lc:LocalConnection = new LocalConnection();
      lc.client = new CustomClient1();
      try
      {
        lc.connect("conn1");
      }
      catch (error:Error)
      {
        trace("error:: already connected");
      }
    }
  }
}
```

우선 이 코드는 `lc`라는 LocalConnection 객체를 만들고 `client` 속성을 사용자 정의 클래스인 CustomClient1로 설정합니다. 다른 Flash Player 인스턴스에서 이 로컬 연결 인스턴스에 있는 메서드를 호출하면 Flash Player가 CustomClient1 클래스에서 해당 메서드를 찾습니다.

Flash Player 인스턴스가 이 SWF 파일에 연결하고 지정된 로컬 연결의 메서드를 호출하려 할 때마다 CustomClient1 클래스로 설정된 `client` 속성에서 지정한 클래스에 요청이 전송됩니다.

```
package
{
  import flash.events.*;
  import flash.system.fscommand;
  import flash.utils.Timer;
  public class CustomClient1 extends Object
  {
    public function doMessage(value:String = ""):void
    {
      trace(value);
    }
    public function doQuit():void
```

```
    {
      trace("quitting in 5 seconds");
      this.close();
      var quitTimer:Timer = new Timer(5000, 1);
      quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
    }
    public function closeHandler(event:TimerEvent):void
    {
      fscommand("quit");
    }
  }
}
```

LocalConnection 서버를 만들려면 `LocalConnection.connect()` 메서드를 호출하고 고유한 연결 이름을 입력하십시오. 지정한 이름의 연결이 이미 있는 경우 객체가 이미 연결되어 있어 연결하지 못했음을 나타내는 ArgumentError 오류가 생성됩니다.

다음 코드 예제에서는 `conn1`이라는 이름으로 새 소켓 연결을 만드는 방법을 보여 줍니다.

```
try
{
  connection.connect("conn1");
}
catch (error:ArgumentError)
{
  trace("Error! Server already exists\n");
}
```

예외

이전 버전의 ActionScript에서는 연결 이름이 이미 사용되고 있는 경우, `LocalConnection.connect()` 메서드에 부울 값이 반환됩니다. ActionScript 3.0에서는 이름이 이미 사용되고 있는 경우에 오류가 생성됩니다.

보조 SWF 파일에서 기본 SWF 파일을 연결하려면 전송하는 LocalConnection 객체에서 새 LocalConnection 객체를 만든 다음 연결 이름 및 실행할 메서드 이름으로 `LocalConnection.send()` 메서드를 호출해야 합니다. 예를 들어, 이전에 만든 LocalConnection 객체에 연결하려면 다음 코드를 사용합니다.

```
sendingConnection.send("conn1", "doQuit");
```

다음 코드는 기존 LocalConnection 객체를 `conn1`이라는 연결 이름으로 연결하고 원격 SWF 파일에서 `doQuit()` 메서드를 호출합니다. 원격 SWF 파일에 매개 변수를 전송하려면 다음 코드 예제에서 보듯이 `send()` 메서드의 메서드 이름 뒤에 추가 인수를 지정합니다.

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

여러 도메인에서 SWF 문서에 연결

특정 도메인에서만 통신할 수 있도록 하려면 LocalConnection 클래스의 `allowDomain()` 또는 `allowInsecureDomain()` 메서드를 호출하고 이 LocalConnection 객체에 액세스할 수 있는 하나 이상의 도메인 목록을 전달합니다.

이전 버전의 ActionScript에서 `LocalConnection.allowDomain()` 및 `LocalConnection.allowInsecureDomain()`은 개발자에 의해 구현되어야 하고 부울 값을 반환해야 하는 콜백 메서드였습니다. ActionScript 3.0에서 `LocalConnection.allowDomain()` 및 `LocalConnection.allowInsecureDomain()`은 기본 제공 메서드입니다. 따라서 개발자는 허용할 도메인 이름 하나 이상을 전달하여 `Security.allowDomain()` 및 `Security.allowInsecureDomain()`처럼 호출만 하면 됩니다.

`LocalConnection.allowDomain()` 및 `LocalConnection.allowInsecureDomain()` 메서드에 전달할 수 있는 특수한 값은 *와 `localhost` 두 가지입니다. 별표 값 (*)은 모든 도메인에서 액세스를 허용합니다. `localhost` 문자열은 로컬로 설치된 SWF 파일에서 이 SWF 파일을 호출할 수 있도록 합니다.

Flash Player 8은 로컬 SWF 파일에 대한 보안 제한 사항을 추가했습니다. 인터넷 액세스가 허용된 SWF 파일이라도 로컬 파일 시스템에 액세스할 수 있는 권한은 없습니다. `localhost`를 지정하면 임의의 로컬 SWF 파일이 SWF 파일에 액세스할 수 있습니다. `LocalConnection.send()` 메서드가 호출 코드가 액세스 권한을 가지고 있지 않은 보안 샌드박스로부터 SWF 파일과 통신하려고 하면 `securityError` 이벤트(`SecurityErrorEvent.SECURITY_ERROR`)가 전달됩니다. 이 오류를 해결하려면 수신자의 `LocalConnection.allowDomain()` 메서드에서 호출자의 도메인을 지정할 수 있습니다.

같은 도메인의 SWF 파일 간에만 통신을 구현하는 경우 밑줄(_)로 시작하지 않고 도메인 이름(예: `myDomain:connectionName`)을 지정하지 않는 `connectionName` 매개 변수를 지정할 수 있습니다. `LocalConnection.connect(connectionName)` 명령에서 같은 문자열을 사용합니다.

다른 도메인의 SWF 파일 간에 통신을 구현하는 경우 밑줄로 시작하는 `connectionName` 매개 변수를 지정합니다. 밑줄을 지정하면 수신 LocalConnection 객체가 있는 SWF 파일을 도메인 간에 보다 자유롭게 이동할 수 있습니다. 다음과 같은 두 가지 경우가 가능합니다.

- `connectionName`의 문자열이 밑줄로 시작하지 않을 경우 Flash Player는 상위 도메인 이름과 콜론이 포함된 접두어(예: `myDomain:connectionName`)를 추가합니다. 이렇게 하면 이름이 같은 다른 도메인의 연결과 충돌하지 않지만 모든 송신 LocalConnection 객체는 이 상위 도메인(예: `myDomain:connectionName`)을 지정해야 합니다. 수신 LocalConnection 객체가 있는 SWF가 다른 도메인으로 이동하면 Flash Player에서 새 상위 도메인이 반영되도록 접두어를 변경합니다(예: `anotherDomain:connectionName`). 모든 송신 LocalConnection 객체는 새 상위 도메인을 가리키도록 수동으로 편집해야 합니다.

- `connectionName`의 문자열이 밑줄(_)로 시작할 경우(예: `_connectionName`) Flash Player는 접두어를 문자열에 추가하지 않습니다. 즉, 수신 및 송신 LocalConnection 객체가 `connectionName`에 동일한 문자열을 사용한다는 것을 의미합니다. 수신 객체에서 `LocalConnection.allowDomain()`을 사용하여 모든 도메인의 연결을 승인하도록 지정할 경우 송신 LocalConnection 객체를 변경하지 않고 수신 LocalConnection 객체가 있는 SWF를 다른 도메인으로 이동할 수 있습니다.

소켓 연결

ActionScript 3.0에는 XML 소켓 연결과 이진 소켓 연결이라는 두 가지 종류의 소켓 연결이 가능합니다. XML 소켓을 사용하면 원격 서버에 연결하고 명시적으로 닫힐 때까지 열린 상태로 유지되는 서버 연결을 만들 수 있습니다. 이를 통해 지속적으로 새로운 서버 연결을 열지 않고도 서버와 클라이언트 간에 XML 데이터를 교환할 수 있습니다. XML 소켓 서버를 사용하는 경우 다른 좋은 점은 사용자가 명시적으로 데이터를 요청할 필요가 없다는 점입니다. 요청 없이 서버에서 데이터를 보낼 수 있으며 XML 소켓 서버에 연결된 모든 클라이언트에 데이터를 보낼 수 있습니다.

클라이언트와 서버에서 특별히 XML 패킷을 교환하지 않아도 된다는 점을 제외하면 이진 소켓 연결도 XML 소켓과 비슷합니다. 대신 연결할 때 이진 정보로 데이터를 전송할 수 있습니다. 따라서 메일 서버(POP3, SMTP 및 IMAP) 및 뉴스 서버(NNTP)를 포함한 다양한 서비스에 연결할 수 있습니다.

Socket 클래스

ActionScript 3.0에서 처음 도입된 Socket 클래스는 ActionScript 코드를 활성화하여 소켓 연결을 만들고 원시 이진 데이터를 읽고 쓸 수 있도록 합니다. XMLSocket 클래스와 비슷하지만 수신되거나 전송되는 데이터의 형식을 지정하지는 않습니다. 소켓 클래스는 이진 프로토콜을 사용하는 서버와 상호 운용하는 데 유용합니다. 이진 소켓 연결을 사용하여 POP3, SMTP, IMAP 및 NNTP 같은 여러 가지 인터넷 프로토콜과 상호 작업할 수 있는 코드를 작성할 수 있습니다. 따라서 Flash Player에서 메일 및 뉴스 서버에 연결할 수 있습니다.

Flash Player에서는 서버의 이진 프로토콜을 사용하여 해당 서버와 직접 통신할 수 있습니다. 일부 서버에서는 big-endian 바이트 순서를 사용하는 반면 일부 서버에서는 little-endian 바이트 순서를 사용합니다. "네트워크 바이트 순서"가 big-endian이므로 인터넷에서 대부분의 서버는 big-endian 바이트 순서를 사용합니다. little-endian 바이트 순서가 널리 사용되는 이유는 Intel® x86 아키텍처에서 사용되기 때문입니다. Endian 바이트 순서를 선택할 때는 데이터를 주고받는 서버의 바이트 순서와 일치하는 바이트 순서를 사용해야 합니다. IDataInput 및 IDataOutput 인터페이스에서 수행되는 모든 연산과 그러한 인터페이스(ByteArray, Socket 및 URLStream)를 구현하는 클래스는 기본적으로 big-endian 형식으로 인코딩됩니다. 즉, 가장 중요한 바이트가 앞에 옵니다. 이는 Java 및 공식 네트워크 바이트 순서와 일치시키기 위한 것입니다. big-endian 바이트 순서를 사용할지 또는 little-endian 바이트 순서를 사용할지 변경하려면 `endian` 속성을 `Endian.BIG_ENDIAN` 또는 `Endian.LITTLE_ENDIAN`으로 설정할 수 있습니다.

팁	소켓 클래스는 IDataInput 및 IDataOutput 인터페이스(flash.utils 패키지에 있음)에서 구현되는 모든 메서드를 상속하며 이러한 메서드는 소켓에서 쓰고 읽는 데 사용해야 합니다.

XMLSocket 클래스

ActionScript는 서버와의 연속적인 연결을 제공하는 내장 XMLSocket 클래스를 제공합니다. 이와 같은 연속 연결은 대기 문제를 해결하며 채팅 응용 프로그램이나 멀티플레이어 게임과 같은 실시간 응용 프로그램에서 주로 사용됩니다. 기존 HTTP 기반 채팅 솔루션은 빈번하게 서버를 폴링하여 HTTP 요청을 사용해 새로운 메시지를 다운로드합니다. 반면에, XMLSocket 채팅 솔루션은 서버와의 연결을 유지하여 클라이언트가 요청하지 않아도 서버에 들어오는 메시지를 서버가 즉시 전송할 수 있도록 합니다.

소켓 연결을 만들려면 소켓 연결 요청을 기다리고 응답을 SWF 파일에 전송하는 서버측 응용 프로그램을 만들어야 합니다. 이 유형의 서버측 응용 프로그램은 Java, Python 또는 Perl과 같은 프로그래밍 언어로 작성할 수 있습니다. XMLSocket 클래스를 사용하려면 XMLSocket 클래스에 사용된 프로토콜을 이해할 수 있는 데몬이 서버 컴퓨터에서 실행되고 있어야 합니다. 다음 목록에 이 프로토콜이 설명되어 있습니다.

- XML 메시지는 전이중 TCP/IP 스트림 소켓 연결을 통해 전송됩니다.
- 각 XML 메시지는 완전한 XML 문서이며 0바이트로 끝납니다.
- 단일 XMLSocket 연결에서 송수신할 수 있는 XML 메시지 수에는 제한이 없습니다.

중요	XMLSocket 클래스는 자동으로 방화벽을 통해 터널링할 수 없습니다. 그 이유는 RTMP(Real-Time Messaging Protocol) 프로토콜과 달리, XMLSocket에는 HTTP 터널링 기능이 없기 때문입니다. HTTP 터널링을 사용해야 할 경우 Flash Remoting 또는 Flash Media Server(RTMP 지원)를 대신 사용하십시오.

XMLSocket 객체가 연결할 수 있는 서버 및 연결 방법에는 다음과 같은 제약이 있습니다.

- XMLSocket.connect() 메서드는 1024 이상의 TCP 포트 번호에만 연결할 수 있습니다. 이 제한 사항 때문에 XMLSocket 객체와 통신하는 서버 데몬 역시 1024 이상의 포트 번호에 할당되어야 합니다. 1024보다 작은 포트 번호는 주로 FTP (21), Telnet (23), SMTP (25), HTTP (80) 및 POP3 (110) 등과 같은 시스템 서비스에 사용되므로 보안 문제 때문에 XMLSocket 객체가 이러한 포트를 사용하지 못하도록 차단한 것입니다. 이렇게 포트 번호를 제한함으로써 이러한 리소스가 부적절하게 액세스되거나 남용될 가능성을 줄일 수 있습니다.
- XMLSocket.connect() 메서드는 SWF 파일이 저장된 동일 도메인의 컴퓨터에만 연결할 수 있습니다. 이 제약은 로컬 디스크에서 실행되는 SWF 파일에는 적용되지 않습니다. (이 제약은 URLLoader.load()의 보안 규칙과 동일합니다.) SWF가 있는 도메인과 다른 도메인에서 실행 중인 서버 데몬에 연결하려면 해당 서버에 특정 도메인으로부터의 액세스를 허용하는 보안 정책 파일을 작성하면 됩니다.

참고	**XMLSocket 객체와 통신할 서버를 설정하는 작업은 그리 간단하지 않습니다. 응용 프로그램에서 실시간 상호 작용이 필요하지 않다면 XMLSocket 클래스 대신 URLLoader 클래스를 사용합니다.**

XMLSocket 클래스의 XMLSocket.connect() 및 XMLSocket.send() 메서드를 사용하여 소켓 연결 상태에서 서버와 XML을 교환할 수 있습니다. XMLSocket.connect() 메서드는 웹 서버 포트와의 소켓 연결을 설정하며 XMLSocket.send() 메서드는 소켓 연결에 지정된 서버에 XML 객체를 전달합니다.

XMLSocket.connect() 메서드를 호출할 때 Flash Player에서는 서버와 TCP/IP 연결을 설정하고 다음 중 하나가 발생할 때까지 이 연결을 그대로 유지합니다.

- XMLSocket 클래스의 XMLSocket.close() 메서드가 호출됩니다.
- XMLSocket 객체에 대한 참조가 더 이상 존재하지 않습니다.
- Flash Player가 종료됩니다.
- 연결이 끊어집니다(예: 모뎀 연결 해제).

Java XML 소켓 서버 만들기 및 연결

다음 코드는 수신 연결을 허용하고 명령 프롬프트 창에 수신된 메시지를 표시하는, Java로 작성된 간단한 XMLSocket 서버를 보여 줍니다. 명령줄에서 서버를 시작할 때 다른 포트 번호를 지정할 수 있지만 기본적으로 새로운 서버는 로컬 컴퓨터의 포트 8080에서 만들어집니다.

새 텍스트 문서를 만들고 다음 코드를 추가합니다.

```
import java.io.*;
import java.net.*;

class SimpleServer
{
  private static SimpleServer server;
  ServerSocket socket;
  Socket incoming;
  BufferedReader readerIn;
  PrintStream printOut;

  public static void main(String[] args)
  {
    int port = 8080;

    try
    {
      port = Integer.parseInt(args[0]);
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      // 예외를 포착하고 계속 진행합니다 .
    }

    server = new SimpleServer(port);
  }

  private SimpleServer(int port)
  {
    System.out.println(">> Starting SimpleServer");
    try
    {
      socket = new ServerSocket(port);
      incoming = socket.accept();
      readerIn = new BufferedReader(new
  InputStreamReader(incoming.getInputStream()));
      printOut = new PrintStream(incoming.getOutputStream());
      printOut.println("Enter EXIT to exit.\r");
      out("Enter EXIT to exit.\r");
      boolean done = false;
      while (!done)
      {
```

```
            String str = readerIn.readLine();
            if (str == null)
            {
              done = true;
            }
            else
            {
              out("Echo: " + str + "\r");
              if(str.trim().equals("EXIT"))
              {
                done = true;
              }
            }
            incoming.close();
          }
        }
        catch (Exception e)
        {
          System.out.println(e);
        }
      }

      private void out(String str)
      {
        printOut.println(str);
        System.out.println(str);
      }
}
```

문서를 하드 디스크에 SimpleServer.java로 저장하고 Java 컴파일러를 사용하여 컴파일하면 SimpleServer.class라는 Java 클래스 파일이 만들어집니다.

명령 프롬프트를 열고 `java SimpleServer`를 입력하여 XMLSocket 서버를 시작할 수 있습니다. SimpleServer.class 파일은 로컬 컴퓨터 네트워크의 어디에나 보관할 수 있고 웹 서버의 루트 디렉터리에만 놓지 않아도 됩니다.

팁 **파일이 Java 클래스 경로에 없어 서버를 시작할 수 없는 경우** `java -classpath . SimpleServer`**로 서버를 시작하십시오.**

ActionScript 응용 프로그램에서 XMLSocket에 연결하려면 다음과 같이 호스트 이름과 포트 번호를 전달할 때 XMLSocket 클래스의 새 인스턴스를 만들고 `XMLSocket.connect()` 메서드를 호출해야 합니다.

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

`XMLSocket.connect()`에 대한 호출이 호출자의 보안 샌드박스 외부에 있는 서버 또는 1024 미만의 포트에 대한 연결을 시도하면 `securityError`(flash.events.SecurityErrorEvent) 이벤트가 발생합니다.

서버에서 데이터를 수신할 때마다 data 이벤트(flash.events.DataEvent.DATA)가 전달됩니다.

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
  trace("[" + event.type + "] " + event.data);
}
```

XMLSocket 서버에 데이터를 전송하려면 XMLSocket.send() 메서드를 사용하고 XML 객체나 문자열을 전달합니다. Flash Player가 제공된 매개 변수를 String 객체로 변환하고 XMLSocket 서버에 내용을 전송하고 뒤에 0바이트를 붙입니다.

```
xmlsock.send(xmlFormattedData);
```

XMLSocket.send() 메서드는 데이터가 성공적으로 전송되었는지 여부를 나타내는 값을 반환하지 않습니다. 데이터를 전송하려 할 때 오류가 발생하면 IOError 오류가 발생합니다.

팁	XML 소켓 서버에 전송하는 각 메시지는 개행(\n) 문자로 끝나야 합니다.

로컬 데이터 저장

"Flash 쿠키"라고도 하는 공유 객체는 방문한 사이트에서 사용자의 컴퓨터에 만들 수 있는 데이터 파일입니다. 예를 들어, 자주 방문하는 웹 사이트의 모양과 느낌을 개인 설정할 수 있도록 허용하는 방식으로 웹 브라우징 환경을 향상시키는 데 공유 객체가 가장 자주 사용됩니다. 공유 객체는 컴퓨터의 데이터에 아무 작업도 할 수 없으며 아무런 관련이 없습니다. 더욱 중요한 것은 사용자가 직접 제공하지 않는 한 공유 객체가 전자 메일 주소나 기타 다른 개인 정보에 액세스할 수 없다는 점입니다.

정적 SharedObject.getLocal() 또는 SharedObject.getRemote() 메서드를 사용하여 새 공유 객체 인스턴스를 만들 수 있습니다. getLocal() 메서드는 현재 클라이언트에서만 사용할 수 있는 로컬의 공유 객체를 로드하려고 하는 반면 getRemote() 메서드는 Flash Media Server와 같은 서버를 통해 여러 클라이언트에서 공유할 수 있는 원격 공유 객체를 로드하려고 합니다. 로컬 공유 객체나 원격 공유 객체가 없으면 getLocal() 및 getRemote() 메서드가 새로운 SharedObject 인스턴스를 만듭니다.

다음 코드는 test라는 로컬 공유 객체를 로드하려고 시도합니다. 이 공유 객체가 없으면 해당 이름으로 새 공유 객체가 만들어집니다.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

test라는 공유 객체가 없으면 0바이트 크기의 새 공유 객체가 만들어집니다. 이전에 공유 객체가 있었다면 이 객체의 현재 크기(바이트)가 반환됩니다.

다음 예제와 같이 데이터 객체에 값을 할당하여 공유 객체에 데이터를 저장할 수 있습니다.

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

test라는 공유 객체와 now 매개 변수가 이미 있다면 기존 값을 덮어씁니다. 다음 예제와 같이 SharedObject.size 속성을 사용하여 공유 객체가 이미 있는지 확인할 수 있습니다.

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // 공유 객체가 없습니다.
  trace("created...");
  so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

앞의 코드는 size 매개 변수를 사용하여 지정한 이름의 공유 객체 인스턴스가 이미 있는지 확인합니다. 다음 코드를 테스트하면 코드를 실행할 때마다 공유 객체가 다시 만들어집니다. 공유 객체를 사용자의 하드 드라이브에 저장하려면 다음 예제와 같이 명시적으로 SharedObject.flush() 메서드를 호출해야 합니다.

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // 공유 객체가 없습니다.
  trace("created...");
  so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

flush() 메서드를 사용하여 공유 객체를 사용자의 하드 드라이브에 기록하는 경우 다음 예제에서 보듯이 Flash Player Settings Manager(www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager07.html)를 사용하여 명시적으로 로컬 저장을 사용하지 않도록 설정했는지 주의하여 확인해야 합니다.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

값은 공유 객체의 data 속성에서 속성 이름을 지정하여 공유 객체에서 가져올 수 있습니다. 예를 들어, 다음 코드를 실행하는 경우 Flash Player에서 SharedObject 인스턴스가 몇 분 전에 만들어졌는지 표시합니다.

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // 공유 객체가 없습니다 .
  trace("created...");
  so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 /
  60).toPrecision(2) + " minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

앞의 코드를 처음 실행하는 경우 최초의 크기가 0바이트이고 이름이 test인 SharedObject 인스턴스가 새로 만들어집니다. 처음에는 크기가 0바이트이므로 if 문에서 true로 평가하고 로컬 공유 객체에 now라는 새 속성을 추가합니다. 현재 시간에서 now 속성의 값을 빼면 공유 객체가 만들어진지 얼마나 되었는지 계산할 수 있습니다. 매번 앞의 코드를 실행한 후에는 공유 객체의 크기가 0보다 커야 하며 코드에서 공유 객체가 몇 분 전에 만들어졌는지 추적합니다.

공유 객체의 내용 표시

값은 data 속성 내에 있는 공유 객체에 저장됩니다. 다음 예제에서 보듯이 for..in 루프를 사용하여 공유 객체 인스턴스의 각 값을 반복할 수 있습니다.

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
  trace(i + ":\t" + so.data[i]);
}
```

보안 SharedObject 만들기

getLocal() 또는 getRemote()를 사용하여 로컬 또는 원격 SharedObject를 만들 때 이 공유 객체에 대한 액세스를 HTTPS 연결을 통해 전달되는 SWF 파일로만 제한할지 결정하는 secure라는 선택 매개 변수가 있습니다. 이 매개 변수를 true로 설정하고 SWF 파일이 HTTPS를 통해 전달되면 Flash Player에서 새 보안 공유 객체를 만들거나 기존의 보안 공유 객체에 대한 참조를 가져옵니다. 이 보안 공유 객체는 보안 매개 변수가 true로 설정된 SharedObject.getLocal()을 호출하는 HTTPS를 통해 전달되는 SWF 파일에 의해서만 읽거나 쓸 수 있습니다. 이 매개 변수를 false로 설정하고 SWF 파일이 HTTPS를 통해 전달되면 Flash Player에서 새 공유 객체를 만들거나 기존의 공유 객체에 대한 참조를 가져옵니다.

이 공유 객체는 HTTPS가 아닌 연결을 통해 제공되는 SWF 파일에 의해 읽거나 쓸 수 있습니다. SWF 파일이 비 HTTPS 연결을 통해 전달되고 이 매개 변수를 true로 설정하려고 하면 새 공유 객체를 만들지 못하고(또는 이전에 만든 보안 공유 객체에 대한 액세스하지 못함), 오류가 발생하며, 공유 객체가 null로 설정됩니다. HTTPS가 아닌 연결에서 다음 코드를 실행하려고 하면 SharedObject.getLocal() 메서드에서 오류를 발생시킵니다.

```
try
{
  var so:SharedObject = SharedObject.getLocal("contactManager", null,
  true);
}
catch (error:Error)
{
  trace("Unable to create SharedObject.");
}
```

만들어진 공유 객체는 이 매개 변수 값에 상관없이 도메인에 허용된 총 디스크 공간에 반영됩니다.

파일 업로드 및 다운로드 작업

FileReference 클래스로 클라이언트와 서버 사이에 파일을 업로드하고 다운로드하는 기능을 추가할 수 있습니다. 사용자에게 Windows 운영 체제의 [열기] 대화 상자와 같이 업로드할 파일 또는 다운로드 위치를 선택하라는 대화 상자가 나타납니다.

ActionScript를 사용하여 만든 FileReference 객체 각각은 사용자의 하드 디스크에서 하나의 파일을 참조합니다. 객체는 파일 크기, 유형, 이름, 만든 날짜 및 수정 날짜에 관한 정보가 들어 있는 속성이 있습니다.

예외	Mac OS에서만 `creator` 속성이 지원됩니다. 다른 모든 플랫폼에서는 `null`을 반환합니다.

FileReference 클래스의 인스턴스를 두 가지 방법으로 만들 수 있습니다. 다음 코드와 같이 `new` 연산자를 사용할 수 있습니다.

```
import flash.net.FileReference;
var myFileReference:FileReference = new FileReference();
```

또는 사용자의 시스템에 사용자에게 업로드할 파일을 하나 이상 선택하라는 내용의 대화 상자를 열고 사용자가 하나 이상의 파일을 선택하면 FileReference 객체의 배열을 만드는 `FileReferenceList.browse()` 메서드를 호출할 수 있습니다. 각각의 FileReference 객체는 대화 상자에서 사용자가 선택한 파일 하나를 나타냅니다. FileReference 객체에는 FileReference 속성(예: `name`, `size` 또는 `modificationDate`)의 데이터가 포함되어 있지 않습니다.

- `FileReference.browse()` 메서드나 `FileReferenceList.browse()` 메서드가 호출되었고 사용자가 파일 선택 대화 상자에서 파일을 선택했습니다.
- `FileReference.download()` 메서드가 호출되었고 사용자가 파일 선택 대화 상자에서 파일을 선택했습니다.

예외	다운로드를 수행하는 경우 다운로드가 완료되기 전에는 `FileReference.name` 속성만 채워집니다. 파일이 다운로드되면 모든 속성을 사용할 수 있습니다.

`FileReference.browse()`, `FileReferenceList.browse()` 또는 `FileReference.download()` 메서드에 대한 호출을 실행할 때 대부분의 플레이어에서는 계속 SWF 파일을 재생합니다.

FileReference 클래스

FileReference 클래스를 사용하면 사용자 컴퓨터와 서버 사이에서 파일을 업로드하거나 다운로드할 수 있습니다. 운영 체제의 대화 상자에는 업로드할 파일이나 다운로드 위치를 선택하라는 메시지가 표시됩니다. FileReference 객체는 사용자 디스크의 단일 파일을 의미하며, 그 속성에는 파일 크기, 유형, 이름, 만든 날짜, 변경 날짜 및 작성자에 대한 정보가 포함됩니다.

FileReference 인스턴스는 다음과 같은 두 가지 방법으로 만들어집니다.

- 다음과 같이 `new` 연산자를 FileReference 생성자와 함께 사용

  ```
  var myFileReference:FileReference = new FileReference();
  ```
- FileReference 객체의 배열을 만드는 `FileReferenceList.browse()`를 호출

SWF 파일은 도메인 간 정책 파일에 지정된 모든 도메인과 자체 도메인에 있는 파일에만 액세스하여 업로드 및 다운로드 작업을 수행할 수 있습니다. 업로드 또는 다운로드를 시작하는 SWF 파일이 서버와 같은 도메인에 있지 않은 경우 정책 파일은 서버에 있어야 합니다.

예외

어디에서나 하나의 대화 상자만 열 수 있으므로 한 번에 하나의 `browse()` 또는 `download()` 액션만 실행할 수 있습니다.

파일 업로드를 처리하는 서버 스크립트에는 다음 요소가 포함된 HTTP `POST` 요청을 사용해야 합니다.

- `multipart/form-data` 값이 있는 `Content-Type`
- `name` 속성이 "`Filedata`"로 설정되고 `filename` 속성이 원본 파일 이름으로 설정된 `Content-Disposition` `FileReference.upload()` 메서드에서 `uploadDataFieldName` 매개 변수의 값을 전달하여 사용자 정의 `name` 속성을 지정할 수 있습니다.
- 파일의 이진 내용

다음은 샘플 HTTP `POST` 요청입니다.

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream
```

```
Test File
------------Ij5aeOaeOKM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5aeOaeOKM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

다음 샘플 HTTP POST 요청은 api_sig, api_key 및 auth_token이라는 세 가지 POST 변수를 전송하고 "photo"의 사용자 정의 업로드 데이터 필드 이름 값을 사용합니다.

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5aeOaeOKM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

서버에 파일 업로드

서버에 파일을 업로드하려면 먼저 사용자가 하나 이상의 파일을 선택할 수 있도록 browse() 메서드를 호출하십시오. 그 다음에 FileReference.upload() 메서드를 호출하면 선택한 파일이 서버에 전송되지 않습니다. 사용자가 FileReferenceList.browse() 메서드를 사용하여 여러 파일을 선택한 경우 Flash Player에서 FileReferenceList.fileList라는 선택한 파일의 배열을 만듭니다. 그러면 FileReference.upload() 메서드를 사용하여 각 파일을 업로드할 수 있습니다.

예외

FileReference.browse() 메서드를 사용하면 하나의 파일만 업로드할 수 있습니다. 사용자가 여러 파일을 업로드할 수 있도록 하려면 FileReferenceList.browse() 메서드를 사용해야 합니다.

기본적으로 해당 시스템의 파일 선택 대화 상자에서 개발자는 FileFilter 클래스를 사용하고 파일 필터 인스턴스의 배열을 browse() 메서드에 전달하여 하나 이상의 사용자 정의 파일 형식 필터를 지정할 수 있지만 사용자는 로컬 컴퓨터에서 모든 파일 형식을 선택할 수 있습니다.

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif,
  *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
  "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

사용자가 파일을 선택하고 해당 시스템의 파일 선택 대화 상자에서 [열기] 버튼을 클릭하면 Event.SELECT 이벤트가 전달됩니다. FileReference.browse() 메서드를 사용하여 업로드할 파일을 선택한 경우 파일을 웹 서버에 전송하려면 다음 코드가 필요합니다.

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
  var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm")
  try
  {
    fileRef.upload(request);
```

```
    }
    catch (error:Error)
    {
      trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

참고 `POST` 또는 `GET` 메서드를 사용하는 변수를 전송하려면 `URLRequest.method` 및 `URLRequest.data` 속성을 사용하여 `FileReference.upload()` 메서드로 서버에 데이터를 전송할 수 있습니다.

`FileReference.upload()` 메서드를 사용하여 파일을 업로드하면 다음 이벤트 중 하나가 전달될 수 있습니다.

- `Event.OPEN`: 업로드 작업이 시작될 때 전달됩니다.
- `ProgressEvent.PROGRESS`: 파일 업로드 작업 중 주기적으로 전달됩니다.
- `Event.COMPLETE`: 파일 업로드 작업이 성공적으로 완료되었을 때 전달됩니다.
- `SecurityErrorEvent.SECURITY_ERROR`: 보안 오류로 인해 업로드에 실패할 때 전달됩니다.
- `HTTPStatusEvent.HTTP_STATUS`: HTTP 오류로 인해 업로드에 실패할 때 전달됩니다.
- `IOErrorEvent.IO_ERROR`: 다음과 같은 이유로 업로드되지 않는 경우에 전달됩니다.
  - Flash Player에서 파일을 읽고, 쓰고 또는 전송하는 중에 입/출력 오류가 발생했습니다.
  - SWF가 인증(예: 사용자 이름 및 암호)이 요구되는 서버에 파일을 업로드하려고 시도했습니다. 업로드하는 동안 Flash Player에서 암호를 입력할 수 있는 방법이 제공되지 않습니다.
  - `url` 매개 변수에 잘못된 프로토콜이 포함되어 있습니다. `FileReference.upload()` 메서드는 HTTP 또는 HTTPS를 사용해야 합니다.

참고 Flash Player에서는 인증이 필요한 서버에 대해 완전한 지원을 제공하지 않습니다. 브라우저에서 실행 중인 즉, 브라우저 플러그인이나 Microsoft ActiveX® 컨트롤을 사용하는 SWF 파일만이 인증 및 다운로드에 필요한 사용자 이름과 암호 입력을 요청하는 대화 상자를 제공할 수 있습니다. 플러그인이나 ActiveX 컨트롤을 사용하는 업로드 또는 독립형이나 외부 플레이어를 사용하는 업로드/다운로드의 경우에는 파일이 전송되지 않습니다.

Flash Player에서 파일 업로드를 허용할 수 있도록 ColdFusion에서 서버 스크립트를 만드는 경우 다음과 비슷한 코드를 사용할 수 있습니다.

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./
  ')#" nameconflict="OVERWRITE" />
```

이 ColdFusion 코드는 Flash Player에서 보낸 파일을 업로드하고 ColdFusion 템플릿과 같은 디렉토리에 저장하여 이름이 같은 파일을 덮어씁니다. 앞의 코드에서는 파일 업로드를 허용하는 데 필요한 최소한의 코드를 보여 줍니다. 실제 업무 환경에서는 이 스크립트를 사용하면 안 됩니다. 데이터 유효성 검사를 추가하여 사용자가 잠재적 위험성이 있는 서버측 스크립트 대신 이미지와 같은 허용된 파일 형식만 업로드하도록 하는 것이 좋습니다.

다음 코드에서는 PHP를 사용한 파일 업로드를 보여 줍니다. 여기에는 데이터 유효성 검사가 포함되어 있습니다. 이 스크립트는 업로드 디렉토리에 업로드된 파일 수를 10개로 제한하고, 200KB 미만의 파일만 업로드하도록 하며, JPEG, GIF 또는 PNG 파일만 파일 시스템에 업로드하고 저장하도록 허용합니다.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
  move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/
  ".$_FILES['Filedata']['name']);
  $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
  if ($type == 1 || $type == 2 || $type == 3)
  {
    rename("./temporary/".$_FILES['Filedata']['name'], "./images/
  ".$_FILES['Filedata']['name']);
  }
  else
  {
    unlink("./temporary/".$_FILES['Filedata']['name']);
  }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
  array_push($files, array('./images/'.$file, filectime('./images/
  '.$file')));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
  $files_to_delete = array_splice($files, 0, count($files) -
  $MAXIMUM_FILE_COUNT);
  for ($i = 0; $i < count($files_to_delete); $i++)
  {
    unlink($files_to_delete[$i][0]);
  }
}
print_r($files);
closedir($directory);
```

```
function sorter($a, $b)
{
  if ($a[1] == $b[1])
  {
    return 0;
  }
  else
  {
    return ($a[1] < $b[1]) ? -1 : 1;
  }
}
?>
```

POST 또는 GET 요청 메서드를 사용하여 스크립트를 업로드할 수 있도록 추가 변수를 전달할 수 있습니다. 추가 POST 변수를 업로드 스크립트에 전송하려면 다음 코드를 사용할 수 있습니다.

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
  var params:URLVariables = new URLVariables();
  params.date = new Date();
  params.ssid = "94103-1394-2345";
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  FileReferenceUpload/fileupload.cfm");
  request.method = URLRequestMethod.POST;
  request.data = params;
  fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

앞의 예제에서 원격 서버측 스크립트에 전달하는 새로운 URLVariables 객체가 만들어집니다. 이전 버전의 ActionScript에서는 쿼리 문자열의 값을 전달하여 서버 업로드 스크립트에 변수를 전달할 수 있습니다. ActionScript 3.0에서는 POST 또는 GET 메서드를 사용하는 데이터를 전달할 수 있는 URLRequest 객체를 사용하여 원격 스크립트에 변수를 전달할 수 있습니다. 그러면 더 큰 데이터 집합을 더 쉽고 간편하게 전달할 수 있습니다. GET 또는 POST 요청 메서드를 사용하여 변수를 전달할지 여부를 지정하려면 URLRequest.method 속성을 각각 URLRequestMethod.GET 또는 URLRequestMethod.POST로 설정할 수 있습니다.

또한 ActionScript 3.0에서는 앞의 예(기본값 Filedata를 Custom1로 대체)와 같이 두 번째 매개 변수를 upload() 메서드로 제공하여 기본 Filedata 업로드 파일 필드 이름을 덮어 쓸 수 있습니다.

기본적으로 Flash Player에서는 `true` 값을 세 번째 매개 변수로 `upload()` 메서드에 전달하여 테스트 업로드를 전송하지 않습니다. 테스트 업로드의 목적은 실제로 파일이 성공적으로 업로드될지 및 서버 인증이 필요한 경우 인증이 될지 여부를 확인하는 것입니다.

예외	현재 테스트 업로드는 Windows 기반 Flash Player에서만 가능합니다.

서버에서 파일 다운로드

사용자가 `request` 및 `defaultFileName`라는 두 가지 매개 변수가 필요한 `FileReference.download()` 메서드를 사용하여 서버에서 파일을 다운로드하도록 할 수 있습니다. 첫 번째 매개 변수는 다운로드할 파일의 URL이 포함된 URLRequest 객체입니다. 두 번째 매개 변수는 선택 항목으로, [파일 다운로드] 대화 상자에 표시되는 기본 파일 이름을 지정할 수 있습니다. 두 번째 매개 변수인 `defaultFileName`을 생략하면 지정된 URL에서 파일 이름이 사용됩니다.

다음 코드는 같은 디렉토리에서 index.xml이라는 파일을 SWF 문서로 다운로드합니다.

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

기본 이름을 index.xml 대신 currentnews.xml로 설정하려면 `defaultFileName` 매개 변수를 다음 코드 예제와 같이 지정하십시오.

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

서버 파일 이름이 직관적이지 않거나 서버에서 생성된 경우 파일 이름을 바꾸면 매우 유용할 수 있습니다. 파일을 직접 다운로드하지 않고 서버측 스크립트를 사용하여 파일을 다운로드할 때 `defaultFileName` 매개 변수를 명시적으로 지정하는 것도 좋습니다. 예를 들어, 전달되는 URL 변수에 따라 특정 파일을 다운로드하는 서버측 스크립트가 있다면 `defaultFileName` 매개 변수를 지정해야 합니다. 그렇지 않으면 다운로드된 파일의 기본 이름이 서버측 스크립트 이름이 됩니다.

서버 스크립트에서 구문 분석할 매개 변수를 URL에 첨부하면 `download()` 메서드를 사용하여 서버에 데이터를 보낼 수 있습니다. 다음 ActionScript 3.0 코드 예제는 ColdFusion 스크립트에 전달되는 매개 변수에 따라 문서를 다운로드합니다.

```
package
{
  import flash.display.Sprite;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.net.URLRequestMethod;
  import flash.net.URLVariables;
```

```
  public class DownloadFileExample extends Sprite
  {
    private var fileToDownload:FileReference;
    public function DownloadFileExample()
    {
      var request:URLRequest = new URLRequest();
      request.url = "http://www.[yourdomain].com/downloadfile.cfm";
      request.method = URLRequestMethod.GET;
      request.data = new URLVariables("id=2");
      fileToDownload = new FileReference();
      try
      {
        fileToDownload.download(request, "file2.txt");
      }
      catch (error:Error)
      {
        trace("Unable to download file.");
      }
    }
  }
}
```

다음 코드는 URL 변수 값에 따라 서버에서 두 파일 중 하나를 다운로드하는 ColdFusion 스크립트인 download.cfm을 보여 줍니다.

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
  <cfcase value="2">
    <cfcontent type="text/plain" file="#ExpandPath('two.txt')#"
  deletefile="No" />
  </cfcase>
  <cfdefaultcase>
    <cfcontent type="text/plain" file="#ExpandPath('one.txt')#"
  deletefile="No" />
  </cfdefaultcase>
</cfswitch>
```

FileReferenceList 클래스

FileReferenceList 클래스를 사용하면 사용자가 서버측 스크립트로 업로드할 하나 이상의 파일을 선택할 수 있습니다. 파일 업로드는 `FileReference.upload()` 메서드로 처리되며 이 메서드는 선택한 파일 각각에 대해 별도로 호출해야 합니다.

다음 코드는 두 가지 FileFilter 개체(`imageFilter` 및 `textFilter`)를 만들고 배열에서 `FileReferenceList.browse()` 메서드로 전달합니다. 그러면 [운영 체제 파일] 대화 상자에 파일 형식에 대한 두 가지 가능한 필터가 표시됩니다.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg,
  *.gif, *.png)", "*.jpg; *.jpeg; *.gif; *.png");
```

```
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
  "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
  var success:Boolean = fileRefList.browse(new Array(imageFilter,
  textFilter));
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
```

FileReferenceList를 사용하면 둘 이상의 파일을 선택할 수 있지만 사용자가 FileReferenceList 클래스를 사용하여 하나 이상의 파일을 선택하고 업로드할 수 있도록 하는 것은 `FileReference.browse()`를 사용하여 파일을 선택하는 것과 같습니다. 여러 개의 파일을 업로드하려면 `FileReference.upload()`를 사용하여 선택한 파일을 각각 업로드해야 합니다.

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm");
  var file:FileReference;
  var files:FileReferenceList = FileReferenceList(event.target);
  var selectedFileArray:Array = files.fileList;
  for (var i:uint = 0; i < selectedFileArray.length; i++)
  {
    file = FileReference(selectedFileArray[i]);
    file.addEventListener(Event.COMPLETE, completeHandler);
    try
    {
      file.upload(request);
    }
    catch (error:Error)
    {
      trace("Unable to upload files.");
    }
  }
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

`Event.COMPLETE` 이벤트는 배열의 각 FileReference 객체에 추가되므로 각 파일의 업로드가 끝나면 Flash Player에서 `completeHandler()` 메서드를 호출합니다.

예제: Telnet 클라이언트 만들기

이 Telnet 예제에서는 Socket 클래스를 사용하여 원격 서버와 연결하고 데이터를 전송하는 기술을 보여 줍니다. 이 예제는 다음과 같은 기술에 대해 설명합니다.

- Socket 클래스를 사용하여 사용자 정의 Telnet 클라이언트 만들기
- ByteArray 객체를 사용하여 원격 서버로 텍스트 보내기
- 원격 서버에서 수신한 데이터 처리

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. Telnet 응용 프로그램 파일은 Samples/Telnet 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
TelnetSocket.mxml	MXML 사용자 인터페이스로 구성된 기본 응용 프로그램 파일입니다.
com/example/programmingas3/Telnet/Telnet.as	원격 서버에 연결, 데이터 송수신 및 표시 등 응용 프로그램에 대한 Telnet 클라이언트 기능을 제공합니다.

Telnet 소켓 응용 프로그램 개요

기본 TelnetSocket.mxml 파일은 전체 응용 프로그램의 UI(사용자 인터페이스)를 만드는 역할을 합니다.

이 파일은 UI 외에도 `login()`과 `sendCommand()`라는 두 가지 메서드를 정의하여 지정한 서버에 사용자를 연결합니다.

다음 코드는 기본 응용 프로그램 파일의 ActionScript를 보여 줍니다.

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
  telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
  console.title = "Connecting to " + serverName.text + ":" +
  portNumber.text;
  console.enabled = true;
}
private function sendCommand():void
{
  var ba:ByteArray = new ByteArray();
  ba.writeMultiByte(command.text + "\n", "UTF-8");
  telnetClient.writeBytesToSocket(ba);
  command.text = "";
}
```

코드의 첫 번째 줄은 사용자 정의 com.example.programmingas.socket 패키지에서 Telnet 클래스를 가져옵니다. 코드의 두 번째 줄은 Telnet 클래스의 인스턴스(telnetClient)를 선언하며 이는 이후 connect() 메서드를 통해 초기화됩니다. 그 다음에 connect() 메서드를 선언하고 앞에서 선언한 telnetClient 변수를 초기화합니다. 이 메서드는 사용자가 지정한 Telnet 서버 이름, Telnet 서버 포트 및 표시 목록의 TextArea 구성 요소에 대한 참조를 전달합니다. 표시 목록은 소켓 서버에서 텍스트 응답을 표시하는 데 사용됩니다. connect() 메서드의 마지막 두 줄은 패널의 title 속성을 설정하고 패널 구성 요소를 사용하도록 설정하여 사용자가 원격 서버에 데이터를 전송할 수 있도록 합니다. 기본 응용 프로그램 파일의 마지막 메서드인 sendCommand()는 사용자의 명령을 원격 서버에 ByteArray 객체로 전송하는 데 사용됩니다.

Telnet 클래스 개요

Telnet 클래스는 원격 Telnet 서버에 연결하고 데이터를 송수신하는 작업을 담당합니다.

Telnet 클래스는 다음과 같은 전용 변수를 선언합니다.

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

첫 번째 변수인 serverURL에는 연결할 사용자 지정 서버 주소가 포함되어 있습니다.

두 번째 변수인 portNumber는 현재 실행 중인 Telnet 서버의 포트 번호입니다. 기본적으로 Telnet 서비스는 포트 23에서 실행됩니다.

세 번째 변수인 socket은 serverURL과 portNumber 변수에서 정의한 서버에 연결을 시도할 Socket 인스턴스입니다.

네 번째 변수인 ta는 Stage의 TextArea 구성 요소 인스턴스에 대한 참조입니다. 이 구성 요소는 원격 Telnet 서버의 응답이나 오류 메시지를 표시하는 데 사용됩니다.

마지막 변수인 state는 Telnet 클라이언트에서 지원하는 옵션을 결정하는 데 사용되는 숫자 값입니다.

앞에서 보듯이 Telnet 클래스의 생성자 함수는 기본 응용 프로그램 파일의 connect() 메서드에서 호출합니다.

Telnet 생성자는 server, port 및 output이라는 세 가지 매개 변수를 사용합니다. server 및 port 매개 변수는 Telnet 서버를 실행하는 서버 이름과 포트 번호를 지정합니다. 마지막 매개 변수인 output은 서버 출력을 사용자에게 표시하는 Stage의 TextArea 구성 요소 인스턴스에 대한 참조입니다.

```
public function Telnet(server:String, port:int, 출력:TextArea)
{
  serverURL = server;
```

```
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
      msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
      socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
      msg(error.message + "\n");
      socket.close();
    }
}
```

소켓에 데이터 기록

소켓 연결에 데이터를 기록하려면 Socket 클래스의 write 메서드(예: `writeBoolean()`, `writeByte()`, `writeBytes()` 또는 `writeDouble()`) 중 하나를 호출한 다음 `flush()` 메서드를 사용하여 출력 버퍼의 데이터를 플러시합니다. Telnet 서버에서는 바이트 배열을 매개 변수로 가져와 출력 버퍼로 전송하는 `writeBytes()` 메서드를 사용하여 소켓 연결에 데이터를 기록합니다. `writeBytesToSocket()` 메서드는 다음과 같습니다.

```
public function writeBytesToSocket(ba:ByteArray):void
{
  socket.writeBytes(ba);
  socket.flush();
}
```

이 메서드는 기본 응용 프로그램 파일의 `sendCommand()` 메서드에 의해 호출됩니다.

소켓 서버의 메시지 표시

소켓 서버에서 메시지를 수신하거나 이벤트가 발생할 때마다 사용자 정의 `msg()` 메서드가 호출됩니다. 이 메서드는 문자열을 Stage의 TextArea에 추가하고 사용자 정의 `setScroll()` 메서드를 호출하여 TextArea 구성 요소가 맨 아래쪽까지 스크롤되도록 합니다. `msg()` 메서드는 다음과 같습니다.

```
private function msg(value:String):void
{
  ta.text += value;
```

```
    setScroll();
}
```

TextArea 구성 요소의 내용을 자동으로 스크롤하지 않은 경우 사용자가 서버의 최신 응답을 보려면 텍스트 영역에서 스크롤 막대를 수동으로 드래그해야 합니다.

TextArea 구성 요소 스크롤

`setScroll()` 메서드에는 TextArea 구성 요소의 내용을 세로로 스크롤하는 ActionScript 한 줄이 포함되어 있으므로 사용자는 반환된 텍스트의 마지막 줄을 볼 수 있습니다. 다음 코드 예제는 `setScroll()` 메서드를 보여 줍니다.

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

이 메서드는 `verticalScrollPosition` 속성을 설정합니다. 이 속성은 현재 표시된 문자의 맨 위 행에 대한 행 번호이며 이 번호를 `maxVerticalScrollPosition` 속성 값으로 설정합니다.

예제: 파일 업로드 및 다운로드

FileIO의 예제는 Flash Player에서 파일 다운로드 및 업로드를 수행하는 기술을 보여 줍니다. 이러한 기술은 다음과 같습니다.

- 사용자의 컴퓨터로 파일 다운로드
- 사용자의 컴퓨터에서 파일 업로드
- 진행 중에 다운로드 취소
- 진행 중에 업로드 취소

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. FileIO 응용 프로그램 파일은 Samples/FileIO 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
FileIO.fla 또는 FileIO.mxml	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/fileio/FileDownload.as	서버에서 파일을 다운로드하는 메서드가 포함된 클래스입니다.
com/example/programmingas3/fileio/FileUpload.as	서버로 파일을 업로드하는 메서드가 포함된 클래스입니다.

FileIO 응용 프로그램 개요

FileIO 응용 프로그램에는 사용자가 Flash Player를 사용하여 파일을 업로드하거나 다운로드할 수 있는 사용자 인터페이스가 포함되어 있습니다. 이 응용 프로그램은 먼저 com.example.programmingas3.fileio 패키지에 있는 사용자 정의 구성 요소인 FileUpload와 FileDownload를 정의합니다. 각 사용자 정의 구성 요소에서 `contentComplete` 이벤트를 전달하고 나면 구성 요소의 `init()` 메서드가 호출되어 ProgressBar 및 Button 구성 요소 인스턴스에 대한 참조를 전달하며, 이를 통해 사용자는 파일의 업로드 또는 다운로드 진행률을 확인하거나 진행 중인 파일의 전송을 취소할 수 있게 됩니다.

FileIO.mxml 파일의 관련 코드는 다음과 같습니다. Flash 버전에서는 Stage에 배치된 구성 요소가 FLA 파일에 포함되어 있으며 이러한 구성 요소의 이름은 이 단계에서 설명하는 Flex 구성 요소의 이름과 같습니다.

```
<example:FileUpload id="fileUpload"
  creationComplete="fileUpload.init(uploadProgress, cancelUpload);" />
<example:FileDownload id="fileDownload"
  creationComplete="fileDownload.init(downloadProgress, cancelDownload);"
  />
```

다음 코드는 진행률 막대와 두 개의 버튼이 있는 [파일 업로드] 패널을 보여 줍니다. 첫 번째 버튼인 `startUpload`는 `FileUpload.startUpload()` 메서드를 호출하고 이 메서드는 `FileReference.browse()` 메서드를 호출합니다. 다음 코드는 [파일 업로드] 패널에 대한 코드를 보여 줍니다.

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
    <mx:Button id="startUpload" label="Upload..."
  click="fileUpload.startUpload();" />
    <mx:Button id="cancelUpload" label="Cancel"
  click="fileUpload.cancelUpload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

이 코드는 스테이지에 ProgressBar 구성 요소 인스턴스와 두 개의 Button 구성 요소 버튼 인스턴스를 배치합니다. 사용자가 [업로드] 버튼(`startUpload`)을 클릭하면 사용자가 원격 서버로 업로드할 파일을 선택할 수 있는 운영 체제 대화 상자가 시작됩니다. 또 다른 버튼인 `cancelUpload`는 사용자가 파일 업로드를 시작할 때 활성화되어 언제든지 파일 전송을 중단할 수 있지만 기본적으로 비활성화되어 있습니다.

[파일 다운로드] 패널에 대한 코드는 다음과 같습니다.

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
```

```
    <mx:Button id="startDownload" label="Download..."
  click="fileDownload.startDownload();" />
    <mx:Button id="cancelDownload" label="Cancel"
  click="fileDownload.cancelDownload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

이 코드는 파일 업로드 코드와 매우 비슷합니다. 사용자가 [다운로드] 버튼(startDownload)을 클릭하면 FileDownload.startDownload() 메서드가 호출되어 FileDownload.DOWNLOAD_URL 변수에 지정된 파일을 다운로드하기 시작합니다. 파일이 다운로드되는 동안 진행률 막대가 업데이트되어 파일의 다운로드 비율(속도)을 보여 줍니다. 사용자는 cancelDownload 버튼을 클릭하여 언제든지 다운로드를 취소할 수 있습니다. 그러면 진행 중인 파일 다운로드가 곧 바로 중지됩니다.

원격 서버에서 파일 다운로드

원격 서버에서 파일을 다운로드하는 작업은 flash.net.FileReference 클래스와 사용자 정의 om.example.programmingas3.fileio.FileDownload 클래스에서 처리합니다. 사용자가 [다운로드] 버튼을 클릭하면 Flash Player가 FileDownload 클래스의 DOWNLOAD_URL 변수에 지정된 파일을 다운로드하기 시작합니다.

다음 코드와 같이 com.example.programmingas3.fileio 패키지의 네 가지 변수를 정의하면 FileDownload 클래스가 시작됩니다.

```
  /**
   * 사용자 컴퓨터에 다운로드할 파일의 URL을 하드 코딩합니다.
   */
  private const DOWNLOAD_URL:String = "http://www.yourdomain.com/
  file_to_download.zip";

  /**
   * 파일 다운로드를 처리할 FileReference 인스턴스를 만듭니다.
   */
  private var fr:FileReference;

  /**
   * 다운로드 ProgressBar 구성 요소에 대한 참조를 정의합니다.
   */
  private var pb:ProgressBar;

  /**
   * 진행 중인 다운로드를 즉시 중지하기 위한 "취소" 버튼에 대한 참조를 정의합니다.
   */
  private var btn:Button;
```

첫 번째 변수인 DOWNLOAD_URL에는 파일의 경로가 포함되어 있으며 사용자가 기본 응용 프로그램 파일에서 [다운로드] 버튼을 클릭하면 사용자의 컴퓨터로 다운로드됩니다.

두 번째 변수인 fr은 FileDownload.init() 메서드에서 초기화되며 원격 파일을 사용자의 컴퓨터로 다운로드하는 FileReference 객체입니다.

마지막 두 변수인 pb와 btn에는 FileDownload.init()에서 초기화하는 Stage의 ProgressBar 및 Button 구성 요소 인스턴스에 대한 참조가 포함되어 있습니다.

FileDownload 구성 요소 초기화

FileDownload 클래스에서 init() 메서드를 호출하면 FileDownload 구성 요소가 초기화됩니다. 이 메서드는 각각 ProgressBar 및 Button 구성 요소 인스턴스를 의미하는 pb와 btn이라는 두 개의 매개 변수를 사용합니다.

init() 메서드에 대한 코드는 다음과 같습니다.

```
/**
 * 구성 요소에 대한 참조를 설정하고 OPEN, PROGRESS 및 COMPLETE 이벤트에 대한
 * 리스너를 추가합니다 .
 */
public function init(pb:ProgressBar, btn:Button):void
{
  // 진행률 막대 및 취소 버튼에 대한 참조를 설정합니다 .
  // 이러한 참조는호출하는 스크립트에서 전달됩니다 .
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

파일 다운로드 시작

사용자가 Stage에서 Download Button 구성 요소 인스턴스를 클릭하면 `startDownload()` 메서드가 파일 다운로드 프로세스를 초기화합니다. 다음 코드는 `startDownload()` 메서드를 보여 줍니다.

```
/**
 * DOWNLOAD_URL 상수에 지정된 파일을 다운로드하기 시작합니다 .
 */
public function startDownload():void
{
  var request:URLRequest = new URLRequest();
  request.url = DOWNLOAD_URL;
  fr.download(request);
}
```

먼저 `startDownload()` 메서드는 새로운 URLRequest 객체를 만들고 대상 URL을 `DOWNLOAD_URL` 변수에서 지정한 값으로 설정합니다. 그 다음에 `FileReference.download()` 메서드가 호출되고 새로 만든 URLRequest 객체가 매개 변수로 전달됩니다. 그러면 운영 체제에서 사용자의 컴퓨터에 요청한 문서를 저장할 위치를 선택하라는 메시지가 표시된 대화 상자를 표시합니다. 사용자가 위치를 선택하면 `open` 이벤트(`Event.OPEN`)가 전달되고 `openHandler()` 메서드가 호출됩니다.

`openHandler()` 메서드는 ProgressBar 구성 요소의 `label` 속성에 대한 텍스트 형식을 설정하고 사용자가 진행 도중 즉시 다운로드를 중지할 수 있는 [취소] 버튼을 활성화합니다. `openHandler()` 메서드는 다음과 같습니다.

```
/**
 * OPEN 이벤트가 전달되면 진행률 막대의 레이블을 변경하고 " 취소 " 버튼을 활성화하여
 * 사용자가 다운로드 작업을 중단할 수 있도록 합니다 .
 * download operation.
 */
private function openHandler(event:Event):void
{
  pb.label = "DOWNLOADING %3%%";
  btn.enabled = true;
}
```

파일의 다운로드 진행률 모니터링

원격 서버에서 사용자의 컴퓨터로 파일을 다운로드할 때 일정 간격으로 progress 이벤트(ProgressEvent.PROGRESS)가 전달됩니다. progress 이벤트가 전달될 때마다 progressHandler() 메서드가 호출되고 Stage의 ProgressBar 구성 요소 인스턴스가 업데이트됩니다. progressHandler() 메서드에 대한 코드는 다음과 같습니다.

```
/**
 * 파일이 다운로드되는 동안 진행률 막대의 상태를 업데이트합니다 .
 */
private function progressHandler(event:ProgressEvent):void
{
  pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

progress 이벤트에는 스테이지에서 ProgressBar 구성 요소를 업데이트하는 데 사용되는 bytesLoaded 및 bytesTotal이라는 두 개의 속성이 있습니다. 이 속성은 사용자에게 파일 다운로드가 얼마나 완료되었고 얼마나 남았는지 알려 줍니다. 사용자는 언제든지 진행률 막대 아래쪽의 [취소] 버튼을 클릭하여 파일 전송을 중지할 수 있습니다.

파일이 다운로드되면 complete 이벤트(Event.COMPLETE)는 사용자에게 파일이 다운로드되었음을 알리고 [취소] 버튼을 비활성화하는 completeHandler() 메서드를 호출합니다. completeHandler() 메서드에 대한 코드는 다음과 같습니다.

```
/**
 * 다운로드가 완료되면 진행률 막대의 레이블을 최종적으로 변경합니다 .
 * 이때 다운로드가 이미 완료되었으므로 " 취소 " 버튼을 비활성화합니다 .
 */
private function completeHandler(event:Event):void
{
  pb.label = "DOWNLOAD COMPLETE";
  btn.enabled = false;
}
```

파일 다운로드 취소

사용자는 언제든지 진행률 막대 아래쪽의 [취소] 버튼을 클릭하여 파일 전송을 중지하고 더 이상 다운로드되지 않도록 중단할 수 있습니다. 다음 예제는 다운로드를 취소할 수 있는 코드입니다.

```
/**
 * 현재 파일 다운로드를 취소합니다 .
 */
public function cancelDownload():void
{
  fr.cancel();
  pb.label = "DOWNLOAD CANCELLED";
```

```
    btn.enabled = false;
  }
```

먼저 이 코드는 파일 전송을 즉시 중지하여 더 이상 데이터를 다운로드하지 않도록 합니다. 다음으로 진행률 막대의 label 속성이 업데이트되어 사용자에게 다운로드가 취소되었음을 알립니다. 마지막으로 파일을 다시 다운로드할 때까지 사용자가 이 버튼을 다시 클릭할 수 없도록 [취소] 버튼을 비활성화합니다.

원격 서버에 파일 업로드

파일 업로드 과정은 파일 다운로드 코드와 매우 비슷합니다. FileUpload 클래스는 다음 코드와 같이 네 개의 동일한 변수를 선언합니다.

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/
  your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

`FileDownload.DOWNLOAD_URL` 변수와 달리 `UPLOAD_URL` 변수에는 사용자의 컴퓨터에서 파일을 업로드할 서버측 스크립트의 URL이 있습니다. 나머지 세 개의 변수는 FileDownload 클래스의 변수와 마찬가지 비헤이비어를 수행합니다.

FileUpload 구성 요소 초기화

FileUpload 구성 요소에는 기본 응용 프로그램에서 호출되는 `init()` 메서드가 있습니다. 이 메서드는 Stage의 ProgressBar 및 Button 구성 요소 인스턴스에 대한 참조인 `pb`와 `btn`이라는 두 개의 매개 변수를 사용합니다. 다음으로 `init()` 메서드는 FileUpload 클래스에서 앞서 정의한 FileReference 객체를 초기화합니다. 마지막으로 이 메서드는 FileReference 객체에 네 개의 이벤트 리스너를 할당합니다. `init()` 메서드에 대한 코드는 다음과 같습니다.

```
public function init(pb:ProgressBar, btn:Button):void
{
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.SELECT, selectHandler);
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

파일 업로드 시작

Stage에서 `FileUpload.startUpload()` 메서드를 호출하는 [업로드] 버튼을 클릭하면 파일 업로드가 초기화됩니다. 이 메서드는 운영 체제에서 원격 서버로 업로드할 파일을 선택하라는 메시지를 나타내는 시스템 대화 상자를 표시하는 FileReference 클래스의 `browse()` 메서드를 호출합니다. 다음 코드는 `startUpload()` 메서드에 대한 코드를 보여 줍니다.

```
public function startUpload():void
{
  fr.browse();
}
```

사용자가 업로드할 파일을 선택하면 `select` 이벤트(`Event.SELECT`)가 전달되고 `selectHandler()` 메서드가 호출됩니다. 우선 `selectHandler()` 메서드는 새로운 URLRequest 객체를 만들고 코드 앞 부분에서 정의한 `UPLOAD_URL` 상수의 값을 `URLRequest.url` 속성으로 설정합니다. 마지막으로 FileReference 객체가 선택된 파일을 지정한 서버측 스크립트에 업로드합니다. `selectHandler()` 메서드에 대한 코드는 다음과 같습니다.

```
private function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest();
  request.url = UPLOAD_URL;
  fr.upload(request);
}
```

FileUpload 클래스의 나머지 코드는 FileDownload 클래스에서 정의한 코드와 같습니다. 사용자가 언제든지 업로드를 종료하려는 경우에는 진행률 막대의 레이블을 설정하고 즉시 파일 전송을 중지하는 [취소] 버튼을 클릭할 수 있습니다. 진행률 막대는 `progress` 이벤트(`ProgressEvent.PROGRESS`)가 전달될 때마다 업데이트됩니다. 마찬가지로 업로드가 완료되면 진행률 막대가 업데이트되어 사용자에게 파일이 업로드되었음을 알려 줍니다. 그런 다음에는 새로 파일 전송을 시작할 때까지 [취소] 버튼이 비활성화됩니다.

제 23 장

클라이언트 시스템 환경

23

이 장에서는 사용자의 시스템과 상호 작용하는 방법에 대해 설명하고 지원되는 기능을 확인하는 방법 및 사용 가능한 경우 사용자 시스템에 설치된 IME(Input Method Editor)를 사용하여 다국어 SWF 파일을 만드는 방법을 보여 줍니다. 또한 응용 프로그램 도메인의 일반적인 사용 방법을 보여 줍니다.

목차

클라이언트 시스템 환경의 기초

클라이언트 시스템 환경 소개

다양한 기능을 갖춘 고급 ActionScript 응용 프로그램을 개발할수록 사용자 운영 체제 정보에 대해 보다 자세히 알고 그 기능에 액세스해야 할 필요성을 느끼게 될 것입니다. 클라이언트 시스템 환경은 flash.system 패키지에 있는 클래스를 모아놓은 것으로 다음과 같은 시스템 레벨 기능의 액세스를 가능하게 합니다.

- SWF를 실행하는 응용 프로그램 및 보안 도메인 확인
- 화면 크기(해상도) 및 mp3 오디오 등의 일부 기능 사용 가능 여부와 같은 사용자 Flash Player의 기능 확인
- IME를 통한 다국어 사이트 구축
- Flash Player 컨테이너(예: HTML 페이지 또는 컨테이너 응용 프로그램)와의 상호 작용
- 클립보드에 정보 저장

flash.system 패키지에는 IMEConversionMode 및 SecurityPanel 클래스도 포함되어 있습니다. 이러한 클래스에는 각각 IME 및 Security 클래스를 사용하는 정적 상수가 들어 있습니다.

일반적인 클라이언트 시스템 환경 작업

이 장에서는 ActionScript를 사용한 클라이언트 시스템 작업 시 일반적으로 수행되는 다음 작업에 대해 설명합니다.

- 응용 프로그램에서 사용 중인 메모리 확인
- 클립보드에 텍스트 복사
- 컴퓨터 성능 확인
  - 화면 해상도, 색상, DPI 및 픽셀 종횡비
  - 운영 체제
  - 오디오 스트리밍, 비디오 스트리밍 및 mp3 재생 지원
  - 설치된 Flash Player이 디버거 버전인지 여부
- 응용 프로그램 도메인 작업
  - 응용 프로그램 도메인 정의
  - 응용 프로그램 도메인으로 SWF 파일 코드 분할
- 응용 프로그램에서의 IME 작업
  - IME 설치 여부 확인
  - IME 변환 모드 확인 및 설정
  - 텍스트 필드에 대해 IME 비활성화
  - IME 변환 시점 감지

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- 운영 체제: 컴퓨터에서 Microsoft Windows, Mac OS X 또는 Linux® 등 모든 기타 응용 프로그램 내에 실행되는 기본 프로그램입니다.
- 클립보드: 복사하거나 잘라낸 텍스트 또는 항목이 일시적으로 저장되는 운영 체제 내 컨테이너로, 이 컨테이너에서 응용 프로그램으로 해당 텍스트 또는 항목을 붙여넣습니다.
- 응용 프로그램 도메인: 각각의 SWF 파일에서 사용되는 클래스를 분류하는 메커니즘으로 SWF 파일에 동일한 이름을 가진 별개의 클래스가 있을 경우 해당 클래스가 서로를 덮어쓰지 않도록 합니다.
- IME(Input Method Editor): 표준 키보드를 통해 복잡한 문자 또는 심볼을 입력하도록 지원하는 프로그램 또는 운영 체제 도구입니다.
- 클라이언트 시스템: 프로그래밍 용어에서 *클라이언트*는 개별 컴퓨터에서 실행되면서 단일 사용자에 의해 사용되는 전체 또는 일부 응용 프로그램을 의미합니다. *클라이언트 시스템*은 사용자 컴퓨터의 기본 운영 체제입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 모든 코드 샘플에는 테스트할 값을 작성하기 위한 해당 `trace()` 함수 호출이 포함되어 있습니다.
이 장의 코드 샘플을 테스트하려면:

1. 빈 Flash 문서를 만듭니다.
2. 타임라인에서 키프레임을 선택합니다.
3. [액션] 패널을 열고 [스크립트] 창에 코드 샘플을 복사합니다.
4. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.
   [출력] 패널에서 코드 샘플의 `trace()` 함수에 대한 결과를 확인합니다.

이후의 코드 샘플 중 일부는 보다 복잡하며 클래스로 작성되어 있습니다. 이러한 예제를 테스트하려면:

1. 빈 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 새 ActionScript 파일을 만들고 Flash 문서와 같은 디렉토리에 저장합니다. 파일 이름은 코드 샘플에 있는 클래스의 이름과 일치해야 합니다. 예를 들어, 코드 샘플에 정의된 클래스 이름이 SystemTest인 경우 SystemTest.as라는 이름을 사용하여 ActionScript 파일을 저장합니다.
3. ActionScript 파일에 코드 샘플을 복사한 후 이 파일을 저장합니다.
4. Flash 문서에서 스테이지 또는 작업 영역의 빈 부분을 클릭하여 문서 속성 관리자를 활성화합니다.

5. 속성 관리자의 [문서 클래스] 필드에 텍스트에서 복사한 ActionScript 클래스 이름을 입력합니다.
6. [컨트롤] > [무비 테스트]를 사용하여 프로그램을 실행합니다.

   [출력] 패널에서 예제 결과를 확인합니다.

예제 코드 샘플 테스트와 관련한 기술에 대해서는 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"에서 자세하게 설명합니다.

System 클래스 사용

System 클래스에는 사용자의 운영 체제와 상호 작용하고 Adobe Flash Player의 현재 메모리 사용 현황을 검색할 수 있는 메서드와 속성이 포함되어 있습니다. System 클래스의 메서드와 속성을 사용하는 경우에도 `imeComposition` 이벤트를 수신하거나, 사용자의 현재 코드 페이지를 사용하여 외부 텍스트 파일을 로드하거나 이러한 파일을 유니코드로 로드하도록 Flash Player에 명령하거나, 사용자 클립보드의 내용을 설정할 수 있습니다.

런타임에 사용자의 시스템에 대한 데이터 가져오기

`System.totalMemory` 속성을 확인하면 Flash Player에서 현재 사용 중인 메모리 양(바이트 단위)을 알 수 있습니다. 이 속성을 사용하면 메모리 사용 현황을 모니터링하고 메모리 레벨 변경 방식에 따라 응용 프로그램을 최적화할 수 있습니다. 예를 들어, 특정 시각 효과로 인해 메모리 사용량이 크게 증가하는 경우 이 효과를 수정하거나 모두 제거할 수 있습니다.

`System.ime` 속성은 현재 설치되어 있는 IME(Input Method Editor)에 대한 참조입니다. 이 속성을 사용하면 `addEventListener()` 메서드를 사용하여 `imeComposition` 이벤트(`flash.events.IMEEvent.IME_COMPOSITION`)를 수신할 수 있습니다.

System 클래스의 세 번째 속성은 `useCodePage`입니다. `useCodePage`를 `true`로 설정하면 Flash Player에서 외부 텍스트 파일을 로드할 플레이어가 실행되고 있는 운영 체제의 기존 코드 페이지를 사용합니다. 이 속성을 `false`로 설정하면 Flash Player에 외부 파일을 유니코드로 해석하도록 지시합니다.

`System.useCodePage`를 `true`로 설정하는 경우 Flash Player를 실행하고 있는 운영 체제의 기존 코드 페이지에 외부 텍스트 파일에 사용된 문자가 포함되어 있어야 텍스트를 표시할 수 있습니다. 예를 들어, 중국어 문자가 포함된 외부 텍스트 파일을 로드하는 경우 영문 Windows 코드 페이지에는 중국어 문자가 없으므로 이 코드 페이지를 사용하는 시스템에는 중국어 문자가 표시되지 않습니다.

모든 운영 체제의 사용자가 SWF 파일에 사용된 외부 텍스트 파일을 볼 수 있도록 하려면 모든 외부 텍스트 파일을 유니코드로 인코딩하고 기본적으로 `System.useCodePage`를 `false`로 설정해야 합니다. 이렇게 하면 Flash Player에서 해당 텍스트를 유니코드로 해석합니다.

클립보드에 텍스트 저장

System 클래스에는 Flash Player에서 지정된 문자열로 사용자 클립보드의 내용을 설정할 수 있는 `setClipboard()`라는 메서드가 포함되어 있습니다. `Security.getClipboard()` 메서드를 사용하면 악의적인 사이트에서 사용자의 클립보드에 마지막으로 복사된 데이터에 액세스할 수 있으므로 보안상의 이유로 이 메서드는 포함되지 않습니다.

다음 코드는 보안 오류가 발생하는 경우 오류 메시지가 사용자의 클립보드에 어떤 방식으로 복사될 수 있는지 보여 줍니다. 오류 메시지는 사용자가 응용 프로그램에 잠재적인 버그를 보고하려는 경우에 유용할 수 있습니다.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
  var errorString:String = "[" + event.type + "] " + event.text;
  trace(errorString);
  System.setClipboard(errorString);
}
```

Capabilities 클래스 사용

Capabilities 클래스를 사용하면 개발자가 어떤 환경에서 SWF 파일을 실행 중인지 확인할 수 있습니다. Capabilities 클래스의 다양한 속성을 사용하면 현재 설치된 Flash Player의 버전뿐 아니라 사용자 시스템의 해상도, 사용자 시스템에서 액세스 가능성 소프트웨어를 지원하는지 여부 및 사용자 운영 체제의 언어를 확인할 수 있습니다.

Capabilities 클래스에서 속성을 확인하면 특정한 사용자 환경에 가장 적합하게 응용 프로그램을 사용자 정의할 수 있습니다. 예를 들어, `Capabilities.screenResolutionX` 및 `Capabilities.screenResolutionY` 속성을 확인하여 사용자 시스템에서 사용 중인 화면 해상도를 확인하고 가장 적합할 것 같은 비디오 크기를 결정할 수 있습니다. 또는 외부 mp3 파일을 로드하기 전에 `Capabilities.hasMP3` 속성을 확인하여 사용자 시스템에서 mp3 재생을 지원하는지 여부를 확인할 수 있습니다.

다음 코드는 클라이언트에서 사용 중인 Flash Player 버전의 구문을 분석할 일반 표현식을 사용합니다.

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
  trace("input: " + result.input);
  trace("platform: " + result[1]);
  trace("majorVersion: " + result[2]);
  trace("minorVersion: " + result[3]);
  trace("buildNumber: " + result[4]);
  trace("internalBuildNumber: " + result[5]);
}
```

```
else
{
  trace("Unable to match RegExp.");
}
```

데이터베이스에 정보를 저장할 수 있도록 사용자의 시스템 기능을 서버측 스크립트로 전송하려면 다음 ActionScript 코드를 사용할 수 있습니다.

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

ApplicationDomain 클래스 사용

ApplicationDomain 클래스의 목적은 ActionScript 3.0 정의 표를 저장하는 것입니다. SWF 파일의 모든 코드는 응용 프로그램 도메인에 존재하도록 정의됩니다. 응용 프로그램 도메인을 사용하여 같은 보안 도메인에 있는 여러 클래스를 구분합니다. 그러면 같은 클래스에 대해 여러 가지 정의가 존재할 수 있으며 자식이 부모의 정의를 다시 사용할 수 있습니다.

Loader 클래스 API를 사용하여 ActionScript 3.0에서 작성한 외부 SWF 파일을 로드하는 경우 응용 프로그램 도메인을 사용할 수 있습니다. ActionScript 1.0 또는 ActionScript 2.0에서 작성한 이미지나 SWF 파일을 로드할 때는 응용 프로그램 도메인을 사용할 수 없습니다. 로드된 클래스에 포함된 모든 ActionScript 3.0 정의는 응용 프로그램 도메인에 저장됩니다. SWF 파일을 로드할 때 LoaderContext 객체의 `applicationDomain` 매개 변수를 `ApplicationDomain.currentDomain`으로 설정하여 Loader 객체와 동일한 응용 프로그램 도메인에 파일이 포함되도록 지정할 수 있습니다. 로드된 SWF 파일을 동일한 응용 프로그램 도메인에 저장하면 해당 클래스에 직접 액세스할 수 있습니다. 이 방법은 다음 예제와 같이 관련된 클래스 이름을 통해 액세스가 가능한 포함된 미디어가 들어 있는 SWF 파일을 로드하려는 경우나 로드된 SWF 파일의 메서드에 액세스하려는 경우에 유용합니다.

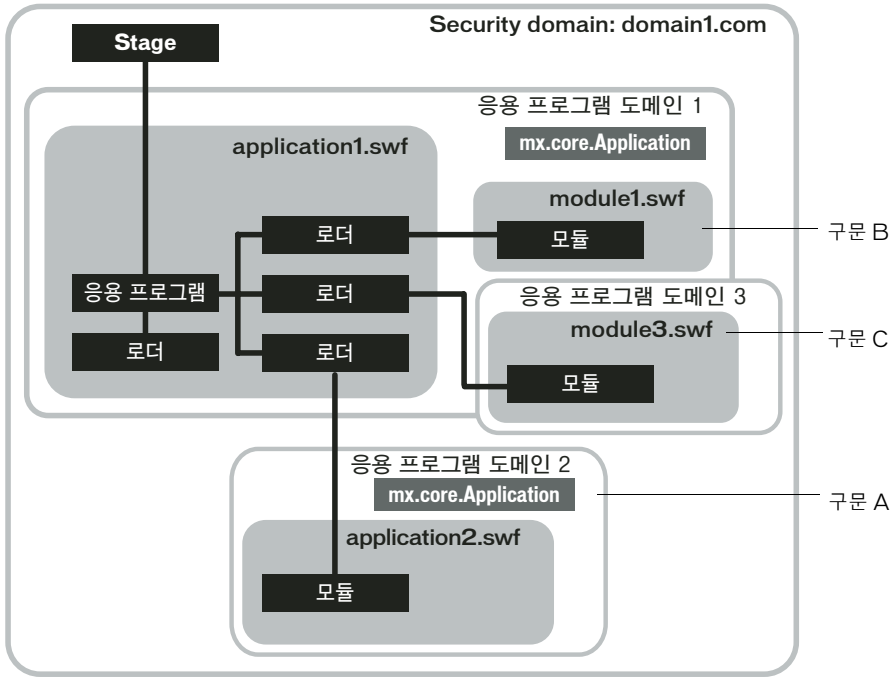
```
package
{
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;
  import flash.system.ApplicationDomain;
  import flash.system.LoaderContext;

  public class ApplicationDomainExample extends Sprite
  {
    private var ldr:Loader;
    public function ApplicationDomainExample()
    {
      ldr = new Loader();
```

```
        var req:URLRequest = new URLRequest("Greeter.swf");
        var ldrContext:LoaderContext = new LoaderContext(false,
    ApplicationDomain.currentDomain);
        ldr.contentLoaderInfo.addEventListener(Event.COMPLETE,
    completeHandler);
        ldr.load(req, ldrContext);
      }
      private function completeHandler(event:Event):void
      {
        ApplicationDomain.currentDomain.getDefinition("Greeter");
        var myGreeter:Greeter = Greeter(event.target.content);
        var message:String = myGreeter.welcome("Tommy");
        trace(message); // Hello, Tommy
      }
    }
}
```

응용 프로그램 도메인을 사용하는 경우 주의해야 할 또 다른 사항은 다음과 같습니다.

- SWF 파일의 모든 코드는 응용 프로그램 도메인에 존재하도록 정의됩니다. *현재 도메인*은 기본 응용 프로그램이 실행되는 위치입니다. *시스템 도메인*에는 현재 도메인을 포함하여 모든 응용 프로그램이 포함되므로 모든 Flash Player 클래스가 들어 있습니다.
- 시스템 도메인을 제외한 모든 응용 프로그램 도메인에는 연결된 부모 도메인이 있습니다. 기본 응용 프로그램 도메인의 부모 도메인은 시스템 도메인입니다. 로드된 클래스는 부모가 이러한 클래스를 아직 정의하지 않은 경우에만 정의됩니다. 로드된 클래스 정의를 새 정의로 재정의할 수는 없습니다.

다음 다이어그램은 단일 도메인인 domain1.com의 다양한 SWF 파일에서 내용을 로드하는 응용 프로그램을 보여 줍니다. 로드한 내용에 따라 서로 다른 응용 프로그램 도메인이 사용될 수 있습니다. 다음에 나오는 텍스트는 응용 프로그램의 각 SWF 파일에 적합한 응용 프로그램 도메인을 설정하는 데 사용되는 논리를 설명합니다.



기본 응용 프로그램 파일은 application1.swf입니다. 여기에는 다른 SWF 파일에서 내용을 로드하는 Loader 객체가 포함되어 있습니다. 이 시나리오에서 현재 도메인은 Application domain 1입니다. 구문 A, 구문 B, 구문 C는 응용 프로그램에서 각 SWF 파일에 적합한 응용 프로그램 도메인을 설정하는 여러 가지 기술을 보여 줍니다.

구문 A: 시스템 도메인의 자식을 만들어 자식 SWF 파일을 구분합니다. 다이어그램에서는 응용 프로그램 도메인 2가 시스템 도메인의 자식으로 만들어집니다. application2.swf 파일이 응용 프로그램 도메인 2에 로드되므로, 해당 클래스 정의는 application1.swf에 정의된 클래스에서 구분됩니다.

이 기술은 이전 응용 프로그램에서 같은 응용 프로그램의 새 버전을 충돌 없이 동적으로 로드하는 데 사용할 수 있습니다. 같은 클래스 이름을 사용하더라도 다른 응용 프로그램 도메인으로 구분되므로 충돌이 없습니다.

다음 코드는 시스템 도메인의 자식인 응용 프로그램 도메인을 만듭니다.

```
request.url = "application2.swf";
request.applicationDomain = new ApplicationDomain();
```

구문 B: 현재 클래스 정의에 새 클래스 정의를 추가합니다. module1.swf의 응용 프로그램 도메인이 현재 도메인(응용 프로그램 도메인 1)으로 설정됩니다. 그러므로 새 클래스 정의를 사용하여 응용 프로그램의 현재 클래스 정의 설정에 추가할 수 있습니다. 이것은 기본 응용 프로그램의 런타임 공유 라이브러리에 사용할 수 있습니다. 로드된 SWF는 RSL(Remote Shared Library)로 처리됩니다. 응용 프로그램이 시작하기 전에 프리로더로 RSL를 로드하려면 이 기술을 사용하십시오.

다음 코드는 응용 프로그램 도메인을 현재 도메인으로 설정합니다.

```
request.url = "module1.swf";
request.applicationDomain = ApplicationDomain.currentDomain;
```

구문 C: 현재 도메인의 새로운 자식 도메인을 만들어 부모 클래스 정의를 사용합니다. module3.swf의 응용 프로그램 도메인은 현재 도메인의 자식이며 자식은 부모 버전의 모든 클래스를 사용합니다. 이 기술은 기본 응용 프로그램 유형을 사용하는 기본 응용 프로그램의 자식으로 로드되는 다중 화면 RIA(Rich Internet Applications)의 모듈로 사용할 수 있습니다. 모든 클래스를 항상 이전 버전과 호환되도록 업데이트하고 로드하는 응용 프로그램이 로드되는 대상보다 항상 새로운 버전인 경우 자식이 부모 버전을 사용합니다. 자식 SWF에 대한 참조를 계속 포함하지 않는 경우 새 응용 프로그램 도메인을 포함하면 가비지 컬렉션의 모든 클래스 정의를 언로드할 수도 있습니다.

이 기술을 사용하면 로드된 모듈이 로더의 단일 객체 및 정적 클래스 멤버를 공유할 수 있습니다.

다음 코드는 현재 도메인의 새 자식 도메인을 만듭니다.

```
request.url = "module3.swf";
request.applicationDomain = new
  ApplicationDomain(ApplicationDomain.currentDomain);
```

IME 클래스 사용

IME 클래스를 사용하면 Flash Player 내에 있는 운영 체제의 IME를 조작할 수 있습니다.

ActionScript를 사용하여 다음을 확인할 수 있습니다.

- IME가 사용자의 컴퓨터에 설치되어 있는지 여부(`Capabilities.hasIME`)
- IME가 사용자의 컴퓨터에서 활성화되어 있는지 아니면 비활성화되어 있는지 여부(`IME.enabled`)
- 현재 IME가 사용하는 변환 모드(`IME.conversionMode`)

입력 텍스트 필드를 특정 IME 컨텍스트에 연결시킬 수 있습니다. 입력 필드 간에 전환할 때 IME를 히라가나(일본어), 최대 폭 숫자, 반폭 숫자, 직접 입력 등으로 서로 전환할 수도 있습니다.

IME를 사용하면 한국어, 중국어 및 일본어 등 ASCII가 아닌 멀티바이트 언어 텍스트 문자를 입력할 수 있습니다.

IME 사용에 대한 자세한 내용은 개발 중인 응용 프로그램을 실행할 운영 체제의 설명서를 참조하십시오. 추가 리소스를 보려면 다음 웹 사이트를 참조하십시오.

- http://www.microsoft.com/globaldev/default.mspx
- http://developer.apple.com/documentation/
- http://java.sun.com/

예외

사용자의 컴퓨터에 IME가 활성화되어 있지 않으면 `Capabilities.hasIME` 이외의 IME 메서드 또는 속성이 호출되지 않습니다. 이때 IME를 수동으로 활성화시키면 ActionScript에서 이후에 IME 메서드 및 속성을 정상적으로 호출할 수 있습니다. 예를 들어, 일본어 IME를 사용하는 경우 IME 메서드 또는 속성을 호출하기 전에 해당 IME를 활성화해야 합니다.

IME 설치 여부 및 활성화 여부 확인

IME 메서드나 속성을 호출하기 전에 현재 사용자의 컴퓨터에 IME가 설치되어 있으며 활성화되어 있는지 항상 확인해야 합니다. 다음 코드는 메서드를 호출하기 전에 IME가 설치 여부 및 활성화 여부를 확인하는 방법을 보여 줍니다.

```
if (Capabilities.hasIME)
{
   if (IME.enabled)
   {
      trace("IME is installed and enabled.");
   }
   else
   {
      trace("IME is installed but not enabled. Please enable your IME and try
   again.");
   }
}
else
{
   trace("IME is not installed. Please install an IME and try again.");
}
```

앞의 코드는 먼저 `Capabilities.hasIME` 속성을 사용하여 IME 설치 여부를 확인합니다. 이 속성이 `true`로 설정되어 있으면 이 코드가 `IME.enabled` 속성을 사용하여 사용자의 IME가 현재 활성화되어 있는지 여부를 확인합니다.

현재 활성화된 IME 변환 모드 확인

다국어 응용 프로그램을 작성할 때 사용자가 현재 활성화되어 있는 변환 모드를 확인해야 할 수 있습니다. 다음 코드는 메서드를 호출하기 전에 IME 설치 여부 확인 방법 및 IME가 설치된 경우 현재 활성화된 IME 변환 모드 확인 방법을 보여 줍니다.

```
if (Capabilities.hasIME)
{
  switch (IME.conversionMode)
  {
    case IMEConversionMode.ALPHANUMERIC_FULL:
      tf.text = "Current conversion mode is alphanumeric (full-width).";
      break;
    case IMEConversionMode.ALPHANUMERIC_HALF:
      tf.text = "Current conversion mode is alphanumeric (half-width).";
      break;
    case IMEConversionMode.CHINESE:
      tf.text = "Current conversion mode is Chinese.";
      break;
    case IMEConversionMode.JAPANESE_HIRAGANA:
      tf.text = "Current conversion mode is Japananese Hiragana.";
      break;
    case IMEConversionMode.JAPANESE_KATAKANA_FULL:
      tf.text = "Current conversion mode is Japanese Katakana (full-
  width).";
      break;
    case IMEConversionMode.JAPANESE_KATAKANA_HALF:
      tf.text = "Current conversion mode is Japanese Katakana (half-
  width).";
      break;
    case IMEConversionMode.KOREAN:
      tf.text = "Current conversion mode is Korean.";
      break;
    default:
      tf.text = "Current conversion mode is " + IME.conversionMode + ".";
      break;
  }
}
else
{
  tf.text = "Please install an IME and try again.";
}
```

앞의 코드는 먼저 IME의 설치 여부를 확인합니다. 다음으로 IMEConversionMode 클래스의 각 상수와 `IME.conversionMode` 속성을 비교하여 현재 IME에서 사용하고 있는 변환 모드를 확인합니다.

IME 변환 모드 설정

IME에서 변환 모드를 설정할 수 없는 경우에 conversionMode 속성을 사용하여 변환 모드를 설정하면 오류가 발생할 수 있으므로 사용자의 IME 변환 모드를 변경할 때 코드가 try..catch 블록에서 래핑되는지 확인해야 합니다. 다음 코드는 IME.conversionMode 속성을 설정할 때 try..catch 블록을 사용하는 방법을 보여 줍니다.

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
  try
  {
    IME.enabled = true;
    IME.conversionMode = IMEConversionMode.KOREAN;
    statusText.text = "Conversion mode is " + IME.conversionMode + ".";
  }
  catch (error:Error)
  {
    statusText.text = "Unable to set conversion mode.\n" + error.message;
  }
}
```

앞의 코드는 먼저 사용자에게 상태 메시지를 표시하는 데 사용되는 텍스트 필드를 만듭니다. 그 다음에는 IME가 설치되어 있는 경우 이 코드가 IME를 활성화하고 변환 모드를 한국어로 설정합니다. 사용자의 컴퓨터에 한국어 IME가 설치되어 있지 않으면 Flash Player에서 오류가 throw되어 try..catch 블록에서 이 오류가 catch됩니다. try..catch 블록은 앞에서 만든 텍스트 필드에 오류 메시지를 표시합니다.

특정 텍스트 필드에 대해 IME 비활성화

경우에 따라 문자를 입력할 때 사용자의 IME를 비활성화할 수 있습니다. 예를 들어, 숫자만 입력할 수 있는 텍스트 필드가 있는 경우 IME를 사용하지 않고 천천히 데이터를 입력할 수 있습니다.

다음 예제는 FocusEvent.FOCUS_IN 및 FocusEvent.FOCUS_OUT 이벤트를 수신하고 그에 따라 사용자의 IME를 비활성화하는 방법을 보여줍니다.

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
```

```
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = false;
  }
}
function focusOutHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = true;
  }
}
```

다음 예제는 phoneTxt와 nameTxt라는 두 가지 입력 텍스트 필드를 만든 다음 두 개의 이벤트 리스너를 phoneTxt 텍스트 필드에 추가합니다. 사용자가 phoneTxt 텍스트 필드를 강조하도록 설정하면 FocusEvent.FOCUS_IN 이벤트가 전달되고 IME가 비활성화됩니다. phoneTxt 텍스트 필드가 강조되지 않으면 FocusEvent.FOCUS_OUT 이벤트가 전달되어 IME를 다시 활성화합니다.

IME 구성 이벤트 수신

구성 문자열을 설정하면 IME 구성 이벤트가 전달됩니다. 예를 들어, 사용자가 IME를 활성화하고 일본어로 문자열을 입력한 경우 사용자가 구성 문자열을 선택하는 즉시 IMEEvent.IME_COMPOSITION 이벤트가 전달됩니다. IMEEvent.IME_COMPOSITION 이벤트를 수신하려면 다음 예제와 같이 이벤트 리스너를 System 클래스 (flash.system.System.ime.addEventListener(...))의 정적 ime 속성에 추가해야 합니다.

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
```

```
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
  IME.enabled = true;
  try
  {
    IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
  }
  catch (error:Error)
  {
    outputTxt.text = "Unable to change IME.";
  }
  System.ime.addEventListener(IMEEvent.IME_COMPOSITION,
  imeCompositionHandler);
}
else
{
  outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
  outputTxt.text = "you typed: " + event.text;
}
```

앞의 코드는 두 개의 텍스트 필드를 만들고 해당 필드를 표시 목록에 추가합니다. 첫 번째 텍스트 필드인 `inputTxt`는 사용자가 일본어 텍스트를 입력할 수 있는 입력 텍스트 필드입니다. 두 번째 텍스트 필드인 `outputTxt`는 사용자에게 오류 메시지를 표시하거나 사용자가 `inputTxt` 텍스트 필드에 입력한 일본어 문자열을 반복하여 표시하는 동적 텍스트 필드입니다.

예제: 시스템 기능 검색

CapabilitiesExplorer 예제는 flash.system.Capabilities 클래스를 사용하여 사용자의 Flash Player에서 지원하는 기능을 확인할 수 있는 방법을 보여 줍니다. 이 예제는 다음과 같은 기술에 대해 설명합니다.

- Capabilities 클래스를 사용하여 사용자의 Flash Player에서 지원하는 기능 검색
- ExternalInterface 클래스를 사용하여 사용자의 브라우저에서 지원하는 브라우저 설정 검색

이 샘플에 대한 응용 프로그램 파일을 구하려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 방문하십시오. CapabilitiesExplorer 응용 프로그램 파일은 Samples/CapabilitiesExplorer 폴더에 있습니다. 이 응용 프로그램은 다음 파일로 구성되어 있습니다.

파일	설명
CapabilitiesExplorer.fla 또는 CapabilitiesExplorer.mxml	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	시스템 Capabilities를 배열에 추가하고, 항목을 정렬하며, ExternalInterface 클래스를 사용하여 브라우저 기능을 검색하는 등 응용 프로그램의 기본 기능을 제공하는 클래스입니다.
capabilities.html	External API와 통신할 수 있도록 필수 JavaScript를 포함하는 HTML 컨테이너입니다.

CapabilitiesExplorer 개요

CapabilitiesExplorer.mxml 파일은 CapabilitiesExplorer 응용 프로그램의 사용자 인터페이스를 설정하는 작업을 담당합니다. 사용자의 Flash Player 기능은 Stage의 DataGrid 구성 요소 인스턴스 내에 표시됩니다. HTML 컨테이너에서 응용 프로그램을 실행하는 경우와 External API를 사용할 수 있는 경우에는 브라우저 기능도 표시됩니다.

기본 응용 프로그램 파일의 `creationComplete` 이벤트가 전달되면 `initApp()` 메서드가 호출됩니다. `initApp()` 메서드는 com.example.programmingas3.capabilities.CapabilitiesGrabber 클래스에서 `getCapabilities()` 메서드를 호출합니다. `initApp()` 메서드에 대한 코드는 다음과 같습니다.

```
private function initApp():void
{
  var dp:Array = CapabilitiesGrabber.getCapabilities();
  capabilitiesGrid.dataProvider = dp;
}
```

CapabilitiesGrabber.getCapabilities() 메서드는 Flash Player 및 브라우저 기능의 정렬된 배열을 반환합니다. 그런 다음 이 배열이 스테이지에 있는 capabilitiesGrid DataGrid 구성 요소 인스턴스의 dataProvider 속성으로 설정됩니다.

CapabilitiesGrabber 클래스 개요

CapabilitiesGrabber 클래스의 정적 getCapabilities() 메서드는 flash.system.Capabilities 클래스에서 배열(capDP)에 각 속성을 추가합니다. 그런 다음 CapabilitiesGrabber 클래스에서 정적 getBrowserObjects() 메서드를 호출합니다. getBrowserObjects() 메서드는 External API를 사용하여 브라우저의 기능이 포함된 브라우저의 navigator 객체를 반복합니다. getCapabilities() 메서드는 다음과 같습니다.

```
public static function getCapabilities():Array
{
  var capDP:Array = new Array();
  capDP.push({name:"Capabilities.avHardwareDisable",
  value:Capabilities.avHardwareDisable});
  capDP.push({name:"Capabilities.hasAccessibility",
  value:Capabilities.hasAccessibility});
  capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
  ...
  capDP.push({name:"Capabilities.version", value:Capabilities.version});
  var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
  if (navArr.length > 0)
  {
    capDP = capDP.concat(navArr);
  }
  capDP.sortOn("name", Array.CASEINSENSITIVE);
  return capDP;
}
```

getBrowserObjects() 메서드는 브라우저의 navigator 객체에서 각 속성의 배열을 반환합니다. 이 배열에 항목 하나 이상의 길이가 포함된 경우 브라우저 기능의 배열(navArr)이 Flash Player 기능(capDP)에 추가되고 전체 배열이 알파벳 순서로 정렬됩니다. 마지막으로 정렬된 배열이 기본 응용 프로그램 파일에 반환된 다음 데이터 격자를 채웁니다.

getBrowserObjects() 메서드에 대한 코드는 다음과 같습니다.

```
private static function getBrowserObjects():Array
{
  var itemArr:Array = new Array();
  var itemVars:URLVariables;
  if (ExternalInterface.available)
  {
    try
    {
      var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
      itemVars = new URLVariables(tempStr);
```

```
        for (var i:String in itemVars)
        {
          itemArr.push({name:i, value:itemVars[i]});
        }
      }
      catch (error:SecurityError)
      {
        // ignore
      }
    }
    return itemArr;
}
```

현재 사용자 환경에서 External API를 사용할 수 있는 경우 Flash Player에서 JavaScript `JS_getBrowserObjects()` 메서드를 호출하여 브라우저의 navigator 객체를 반복하고 URL 인코딩된 값의 문자열을 ActionScript로 반환합니다. 그러면 이 문자열이 URLVariables 객체(`itemVars`)로 변환되고 호출 스크립트로 반환되는 `itemArr` 배열에 추가됩니다.

JavaScript로 통신

CapabilitiesExplorer 응용 프로그램을 구축하는 마지막 단계는 필요한 JavaScript를 작성하여 브라우저의 navigator 객체에서 각 항목을 반복하고 이름-값 쌍을 임시 배열에 추가하는 것입니다. container.html 파일의 JavaScript `JS_getBrowserObjects()` 메서드에 대한 코드는 다음과 같습니다.

```
<script language="JavaScript">
  function JS_getBrowserObjects()
  {
    // 브라우저의 각 항목을 포함할 배열을 만듭니다 .
    var tempArr = new Array();

    // 브라우저의 각 navigator 객체 항목에 대해 반복합니다 .
    for (var name in navigator)
    {
      var value = navigator[name];

      // 현재 값이 string 또는 Boolean 객체이면 배열에 추가하고 ,
      // 그렇지 않으면 항목을 무시합니다 .
      switch (typeof(value))
      {
        case "string":
        case "boolean":

          // 배열에 추가될 임시 문자열을 만듭니다 .
          // JavaScript 의 escape() 함수를 사용하여 값을 URL 인코딩해야 합니다 .
          var tempStr = "navigator." + name + "=" + escape(value);
          // URL 인코딩한 이름 / 값 쌍을 배열로 이동합니다 .
          tempArr.push(tempStr);
```

```
            break;
        }
    }
    // 브라우저의 각 screen 객체 항목에 대해 반복합니다 .
    for (var name in screen)
    {
        var value = screen[name];

        // 현재 값이 숫자이면 배열에 추가하고 , 그렇지 않으면 항목을 무시합니다 .
        switch (typeof(value))
        {
            case "number":
                var tempStr = "screen." + name + "=" + escape(value);
                tempArr.push(tempStr);
                break;
        }
    }
    // 배열을 URL 인코딩된 이름 - 값 쌍의 문자열로 반환합니다 .
    return tempArr.join("&");
  }
</script>
```

이 코드는 navigator 객체에서 모든 이름-값 쌍을 포함하는 있는 임시 배열을 만드는 것으로 시작합니다. 그 다음은 `for..in` 루프를 사용하여 navigator 객체를 반복하고 현재 값의 데이터 유형을 평가하여 원치 않는 값을 제외합니다. 이 응용 프로그램에서는 String 또는 부울 값만 사용하고 다른 데이터 유형(함수 또는 배열)은 무시합니다. navigator 객체의 각 String 또는 부울 값은 `tempArr` 배열에 추가됩니다. 그 다음에는 `for..in` 루프를 사용하여 브라우저의 screen 객체가 반복되고 각 숫자 값이 `tempArr` 배열에 추가됩니다. 마지막으로 임시 배열은 `Array.join()` 메서드를 사용하여 문자열로 변환됩니다. 배열은 앰퍼샌드(&)를 구분 기호로 사용합니다. 그러므로 ActionScript에서 URLVariables 클래스를 사용하여 쉽게 데이터의 구문을 분석할 수 있습니다.

제 24 장

24 인쇄

Adobe® Flash® Player 9는 운영 체제의 인쇄 인터페이스와 통신할 수 있으므로 페이지를 인쇄 스풀러로 전달할 수 있습니다. Flash Player에서 스풀러에 전송하는 각 페이지에는 데이터베이스 값 및 동적 텍스트 등 사용자에게 보이거나, 동적이거나, 스크린 밖에 표시되는 내용이 포함될 수 있습니다. 또한 Flash Player는 사용자의 프린터 설정에 따라 flash.printing.PrintJob 클래스의 속성을 설정하므로 그에 맞게 페이지 형식을 지정할 수 있습니다.

이 장에서는 flash.printing.PrintJob 클래스 메서드와 속성을 사용하여 인쇄 작업을 만들고, 사용자의 인쇄 설정을 읽고, Flash Player 및 사용자 운영 체제의 피드백에 따라 인쇄 작업을 조정하는 전략에 대해 설명합니다.

목차

인쇄의 기초

인쇄 작업 소개

ActionScript 3.0에서는 PrintJob 클래스를 사용하여 표시 내용의 스냅샷을 만들고 출력에 잉크 용지 품질의 표현으로 변환합니다. 인쇄 내용 설정은 여러 요소를 배치하고 크기를 조정하며 원하는 레이아웃을 만드는 등 몇 가지 면에서 화면 표시를 설정하는 것과 동일합니다. 하지만 인쇄가 화면 레이아웃과 구별되는 몇 가지 특성이 있습니다. 예를 들어, 프린터와 컴퓨터 모니터에서 사용되는 해상도가 다르고, 컴퓨터 화면 내용은 동적이고 변화 가능한 반면 인쇄 내용은 본질적으로 정적이며, 인쇄 계획 시에는 페이지 크기가 고정되어 있다는 제약 및 여러 페이지 인쇄 기능을 고려해야 합니다.

이러한 차이는 명백해 보이기는 하지만 ActionScript에서 인쇄 설정 시 반드시 유념해야 합니다. 정확한 인쇄를 위해서는 사용자가 지정하는 값과 사용자 프린터의 특성이 적절하게 조합되어야 하므로 PrintJob 클래스에는 고려해야 할 프린터의 중요 특성을 결정할 수 있는 속성이 포함됩니다.

일반적인 인쇄 작업

이 장에서는 다음과 같은 일반적인 인쇄 작업에 대해 설명합니다.

- 인쇄 작업 시작
- 인쇄 작업에 페이지 추가
- 사용자의 인쇄 작업 취소 여부 결정
- 비트맵 또는 벡터 렌더링 사용 여부 지정
- 페이지 크기, 배율 및 방향 설정
- 인쇄 가능 영역 지정
- 화면 크기를 페이지 크기로 변환
- 여러 페이지 인쇄 작업 수행

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- 스풀러: 인쇄 대기 중인 페이지를 추적하여 프린터가 사용 가능할 때 프린터로 작업을 전송하는 프린터 드라이버 소프트웨어 또는 운영 체제의 일부입니다.
- 페이지 방향 설정: 인쇄할 내용을 용지의 가로 또는 세로 방향으로 회전하는 것입니다.
- 인쇄 작업: 1회의 출력을 구성하는 페이지 또는 페이지 집합입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플 중 상당수는 전체 인쇄 작업 예제 또는 값 확인용 코드가 아니라 코드의 일부입니다. 예제를 테스트하려면 인쇄할 요소를 만들고 코드 샘플에 이러한 요소를 사용합니다. 마지막 두 장에는 인쇄 작업을 수행할 뿐만 아니라 인쇄할 내용을 정의하는 코드가 포함된 전체 인쇄 예제가 나와 있습니다.

예제 코드 샘플을 테스트하려면:

1. 새 Flash 문서를 만듭니다.
2. 타임라인의 프레임 1에서 키프레임을 선택하고 [액션] 패널을 엽니다.
3. [스크립트] 창에 코드 샘플을 복사합니다.
4. 주 메뉴에서 [컨트롤] > [무비 테스트]를 선택하여 SWF 파일을 만들고 예제를 테스트합니다.

페이지 인쇄

인쇄 작업을 처리하려면 PrintJob 클래스의 인스턴스를 사용합니다. Flash Player를 통해 기본 페이지를 인쇄하려면 다음 4개의 명령문을 차례로 사용하십시오.

- `new PrintJob()`: 지정한 이름의 새로운 인쇄 작업 인스턴스를 만듭니다.
- `PrintJob.start()`: 운영 체제의 인쇄 프로세스를 초기화하여 사용자에게 [인쇄] 대화상자를 호출하고 인쇄 작업의 읽기 전용 속성이 선택되도록 합니다.
- `PrintJob.addPage()`: Sprite 객체 및 그 안에 포함된 자식, 인쇄 영역의 크기, 프린터에서 이미지를 벡터로 인쇄할지 또는 비트맵으로 인쇄할지 여부 등 인쇄 작업 내용에 대한 정보가 들어 있습니다. `addPage()`에 대해 연속 호출을 사용하여 여러 페이지에 걸쳐 여러 스프라이트를 인쇄할 수 있습니다.
- `PrintJob.send()`: 페이지를 운영 체제의 프린터로 전송합니다.

그러므로 예를 들면 매우 간단한 인쇄 작업 스크립트가 다음과 같이 보일 수 있습니다(컴파일을 위한 `package`, `import` 및 `class` 문 포함).

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;

  public class BasicPrintExample extends Sprite
  {
    var myPrintJob:PrintJob = new PrintJob();
    var mySprite:Sprite = new Sprite();

    public function BasicPrintExample()
```

```
    {
      myPrintJob.start();
      myPrintJob.addPage(mySprite);
      myPrintJob.send();
    }
  }
}
```

예외

이 예는 인쇄 작업 스크립트의 기본 요소를 보여 주기 위한 것으로 오류 처리는 포함되어 있지 않습니다. 인쇄 작업을 취소하는 사용자에게 적절히 응답하는 스크립트를 작성하려면 672페이지의 "예외 및 반환 처리"를 참조하십시오.

특정 이유로 인해 PrintJob 객체의 속성을 제거할 필요가 있다면 PrintJob 변수를 `null`로 설정하십시오(예: `myPrintJob = null`).

Flash Player 작업 및 시스템 인쇄

Flash Player에서는 운영 체제의 인쇄 인터페이스에 페이지를 전달하므로 Flash Player에서 관리하는 작업의 범위와 운영 체제의 자체 인쇄 인터페이스에서 관리하는 작업의 범위를 알아야 합니다. Flash Player는 인쇄 작업을 초기화하고, 프린터의 페이지 설정 일부를 읽고, 인쇄 작업의 내용을 운영 체제에 전달하고, 사용자나 시스템에서 인쇄 작업을 취소했는지 여부를 확인합니다. 프린터별 대화 상자를 표시하거나, 스풀링된 인쇄 작업을 취소하거나, 프린터의 상태를 보고하는 등 그 밖의 프로세스는 모두 운영 체제에서 처리됩니다. Flash Player에서는 인쇄 작업을 초기화하거나 형식을 지정하는 데 문제가 있는 경우에 응답은 할 수 있지만 운영 체제의 인쇄 인터페이스에서 특정 속성이나 조건에 대해서만 다시 보고할 수 있습니다. 개발자의 경우 코드에 이러한 속성이나 조건에 응답할 수 있는 기능이 있어야 합니다.

예외 및 반환 처리

사용자가 인쇄 작업을 취소한 경우 `addPage()` 및 `send()` 호출을 실행하기 전에 `PrintJob.start()` 메서드가 `true`를 반환하는지 확인해야 합니다. 계속 진행하기에 앞서 이러한 메서드가 취소되었는지 확인하는 간단한 방법은 `if` 문에서 해당 메서드를 다음과 같이 래핑하는 것입니다.

```
if (myPrintJob.start())
{
  // 여기에 addPage() 및 send() 문을 입력하십시오.
}
```

`PrintJob.start()`가 `true`이면 사용자가 [인쇄]를 선택했거나 Flash Player에서 [인쇄] 명령을 초기화했고 `addPage()` 및 `send()` 메서드를 호출할 수 있음을 의미합니다.

또한 인쇄 프로세스를 관리할 수 있도록 이제 Flash Player에서 PrintJob.addPage() 메서드에 대한 예외를 발생시키므로 오류를 포착하고 사용자에게 정보와 옵션을 제공할 수 있습니다. PrintJob.addPage() 메서드가 실패하면 다른 함수를 호출하거나 현재 인쇄 작업을 중지할 수도 있습니다. 다음 예제와 같이 try..catch 문 내에 addPage() 호출을 포함하여 이러한 예외를 포착합니다. 예제에서 [params]는 실제 인쇄할 내용을 지정하는 매개 변수의 자리 표시자입니다.

```
if (myPrintJob.start())
{
  try
  {
    myPrintJob.addPage([params]);
  }
  catch (error:Error)
  {
    // 오류를 처리합니다 .
  }
  myPrintJob.send();
}
```

인쇄 작업이 시작되면 PrintJob.addPage()를 사용하여 내용을 추가할 수 있으며 그로 인해 예외가 생성되는지 여부(예: 사용자가 인쇄 작업을 취소했는지 여부)를 확인할 수 있습니다. 예외가 발생한 경우 catch 문에 논리를 추가하여 사용자(또는 Flash Player)에게 정보와 옵션을 제공하거나 현재 인쇄 작업을 중지할 수 있습니다. 페이지를 제대로 추가하면 계속해서 PrintJob.send()를 사용하여 페이지를 프린터로 전송할 수 있습니다.

Flash Player에서 프린터에 인쇄 작업을 보내는 데 문제가 발생하는 경우, 예를 들어, 프린터가 오프라인인 경우 이 예외도 포착하여 사용자나 Flash Player에 정보나 추가 옵션(예: 메시지 텍스트 표시 또는 Flash 애니메이션 내에 경고 표시)을 제공할 수 있습니다. 예를 들어, 다음 코드와 같이 if..else 문의 텍스트 필드에 새 텍스트를 지정할 수 있습니다.

```
if (myPrintJob.start())
{
  try
  {
    myPrintJob.addPage([params]);
  }
  catch (error:Error)
  {
    // 오류를 처리합니다 .
  }
  myPrintJob.send();
}
else
{
  myAlert.text = "Print job canceled";
}
```

작업 예제는 679페이지의 “예제: 배율 조절, 자르기 및 자동 맞춤”을 참조하십시오.

페이지 속성을 사용한 작업

사용자가 [인쇄] 대화 상자에서 [확인]을 클릭하고 `PrintJob.start()`가 `true`를 반환하면 프린터 설정에서 정의한 속성에 액세스할 수 있습니다. 이러한 속성에는 용지 폭, 용지 높이(`pageHeight` 및 `pageWidth`) 및 용지의 내용 방향 등이 있습니다. 이러한 속성은 프린터 설정이므로 Flash Player에서 조절할 수 없으며 이러한 설정은 변경할 수 없습니다. 그러나 이러한 속성을 사용하여 프린터로 전송할 내용을 현재 설정에 맞게 정렬할 수 있습니다. 자세한 내용은 675페이지의 "크기, 배율 및 방향 설정"을 참조하십시오.

벡터 또는 비트맵 렌더링 설정

각 페이지를 벡터 그래픽이나 비트맵 이미지로 스풀링하도록 인쇄 작업을 수동으로 설정할 수 있습니다. 경우에 따라 벡터 인쇄는 비트맵 인쇄보다 더 작은 스풀 파일을 만들고 더 나은 이미지를 만듭니다. 그러나 내용에 비트맵 이미지가 있는 경우 알파 투명도나 색상 효과를 보존하려면 페이지를 비트맵 이미지로 인쇄해야 합니다. 또한 PostScript 프린터가 아닌 프린터에서는 자동으로 벡터 그래픽을 비트맵 이미지로 변환합니다. 다음과 같이 `printAsBitmap` 매개 변수를 `true`로 설정한 상태로 PrintJobOptions 객체를 전달하여 `PrintJob.addPage()`의 세 번째 매개 변수에서 비트맵 인쇄를 지정합니다.

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

세 번째 매개 변수의 값을 지정하지 않으면 인쇄 작업에서 기본값(벡터 인쇄)을 사용합니다.

예외 `printArea`**(두 번째 매개 변수) 값을 지정하지 않고 비트맵 인쇄 값을 지정하려면** `printArea`**에** `null`**을 사용합니다.**

인쇄 작업 시간 제한 명령문

ActionScript 3.0에서는 이전 버전의 ActionScript와 달리 PrintJob 객체를 단일 프레임으로 제한하지 않습니다. 그러나 사용자가 [인쇄] 대화 상자에서 [확인] 버튼을 클릭하면 운영 체제에서 사용자에게 인쇄 상태 정보를 표시하므로 `PrintJob.addPage()` 및 `PrintJob.send()`를 최대한 빠르게 호출하여 페이지를 스풀러로 보내야 합니다. `PrintJob.send()` 호출이 들어 있는 프레임에 늦게 도달하면 인쇄 과정이 지연될 수 있습니다.

ActionScript 3.0에서는 15초의 스크립트 시간 초과 제한이 있습니다. 그러므로 연속된 인쇄 작업에서 각 기본 명령문 사이의 시간은 15초를 초과할 수 없습니다. 즉, 다음 간격에 15초 스크립트 시간 초과 제한이 적용됩니다.

- `PrintJob.start()`와 첫 번째 `PrintJob.addPage()` 사이
- `PrintJob.addPage()`와 다음 `PrintJob.addPage()` 사이
- 마지막 `PrintJob.addPage()`와 `PrintJob.send()` 사이

이러한 간격 중 하나가 15초를 초과하는 경우 다음에 PrintJob 인스턴스에서 `PrintJob.start()`를 호출하면 `false`가 반환되고 다음에 PrintJob 인스턴스에서 `PrintJob.addPage()`를 반환하면 Flash Player에서 런타임 예외가 발생합니다.

크기, 배율 및 방향 설정

671페이지의 "페이지 인쇄" 섹션에는 기본 인쇄 작업 단계가 자세히 설명되어 있습니다. 지정한 스프라이트의 화면 크기와 위치에 해당하는 인쇄 내용이 출력에 직접 반영됩니다. 그러나 프린터에서 인쇄할 때 다른 해상도를 사용하고, 인쇄된 스프라이트의 모양에 영향을 주는 설정이 있을 수 있습니다.

Flash Player는 운영 체제의 인쇄 설정을 읽을 수 있지만 이러한 속성은 읽기 전용입니다. 인쇄 설정 값에 응답할 수는 있지만 설정할 수는 없습니다. 그러므로 예를 들어, 프린터의 용지 크기 설정을 알아 보고 그 크기에 맞게 내용을 조정할 수 있습니다. 또한 프린터의 여백 설정과 페이지 방향도 확인할 수 있습니다. 프린터 설정에 응답하려면 인쇄 영역을 지정하거나, 화면 해상도와 프린터의 포인트 측정 단위 간에 차이를 조정하거나, 내용을 사용자 프린터의 크기나 방향 설정에 맞게 변형해야 할 수 있습니다.

인쇄 영역에 사각형 사용

`PrintJob.addPage()` 메서드를 사용하면 인쇄할 스프라이트의 영역을 지정할 수 있습니다. 두 번째 매개 변수인 `printArea`는 Rectangle 객체의 양식에 있습니다. 이 매개 변수에 값을 제공하는 옵션은 다음 세 가지입니다.

- 다음 예제와 같이 특정 속성이 있는 Rectangle 객체를 만든 다음 `addPage()` 호출에서 해당 사각형을 사용합니다.

  ```
  private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
  myPrintJob.addPage(sheet, rect1);
  ```

- 아직 Rectangle 객체를 지정하지 않은 경우 다음 예제와 같이 호출 자체 내에서 지정할 수 있습니다.

  ```
  myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
  ```

- `addPage()` 호출에서 세 번째 매개 변수의 값은 제공할 계획이지만 사각형을 지정하지 않으려면 다음과 같이 두 번째 매개 변수에 `null`을 사용할 수 있습니다.

  ```
  myPrintJob.addPage(sheet, null, options);
  ```

예외	**인쇄 크기에 사각형을 지정하려는 경우 `flash.display.Rectangle` 클래스를 가져와야 합니다.**

포인트와 픽셀 비교

사각형의 폭과 높이는 픽셀 값입니다. 프린터에서는 인쇄 측정 단위로 포인트를 사용합니다. 포인트는 고정된 실제 크기(1/72인치)이지만 화면상 픽셀의 크기는 특정 화면의 해상도에 따라 다릅니다. 픽셀과 포인트 사이의 변환 비율은 프린터 설정 및 스프라이트의 배율 조절 여부에 따라 다릅니다. 폭이 72픽셀이고 배율이 조절되지 않은 스프라이트는 폭이 1인치로 인쇄되며 이때 1포인트는 화면 해상도에 관계없이 1픽셀과 같습니다.

인치나 센티미터를 트윕 또는 포인트(1트윕은 1/20포인트)로 변환할 때 다음 등식을 사용할 수 있습니다.

- 1포인트 = 1/72인치 = 20트윕
- 1인치 = 72포인트 = 1440트윕
- 1센티미터 = 567트윕

`printArea` 매개 변수가 생략되거나 잘못 전달되면 스프라이트의 전체 영역이 인쇄됩니다.

배율 조절

인쇄하기 전에 Sprite 객체의 배율을 조절하려면 `PrintJob.addPage()` 메서드를 호출하기 전에 배율 속성(378페이지의 "객체 크기 조작 및 크기 조절" 참조)을 설정하고 인쇄 후 속성을 원래 값으로 다시 설정합니다. Sprite 객체의 배율은 `printArea` 속성과 관계가 없습니다. 즉, 인쇄 영역을 50 x 50픽셀로 지정하면 2500픽셀이 인쇄됩니다. Sprite 객체의 배율을 조절하면 동일하게 2500픽셀이 인쇄되지만 Sprite 객체는 배율이 조절된 크기로 인쇄됩니다.

예제는 679페이지의 "예제: 배율 조절, 자르기 및 자동 맞춤"을 참조하십시오.

가로 또는 세로 방향으로 인쇄

Flash Player에서 방향 설정을 감지할 수 있으므로 다음 예제와 같이 ActionScript에 논리를 작성하여 프린터 설정에 맞게 현재 크기나 회전을 조절할 수 있습니다.

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
  mySprite.rotation = 90;
}
```

참고	용지에서 내용 방향에 대한 시스템 설정을 읽으려는 경우 다음을 사용하여 PrintJobOrientation 클래스를 가져와야 합니다. `import flash.printing.PrintJobOrientation;` PrintJobOrientation 클래스는 페이지에서 내용 방향을 정의하는 상수 값을 제공합니다.

페이지 높이와 폭에 자동 맞춤

프린터 방향 설정을 처리하는 것과 비슷한 방법을 사용하면 `if` 문에 일부 논리를 포함함으로써 페이지 높이와 폭 설정을 읽은 후 그에 맞게 조정할 수 있습니다. 다음은 예제 코드입니다.

```
if (mySprite.height > myPrintJob.pageHeight)
{
  mySprite.scaleY = .75;
}
```

뿐만 아니라 다음 예제와 같이 페이지와 용지의 크기를 비교하여 페이지의 여백 설정을 확인할 수 있습니다.

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

예제: 여러 페이지 인쇄

두 페이지 이상의 내용을 인쇄할 때 각 페이지의 내용을 다른 스프라이트(이 예제에서는 `sheet1`과 `sheet2`)와 연결한 다음 각 스프라이트에 대해 `PrintJob.addPage()`를 사용할 수 있습니다. 다음 코드는 이 기술을 보여 줍니다.

```
package
{
  import flash.display.MovieClip;
  import flash.printing.PrintJob;
  import flash.printing.PrintJobOrientation;
  import flash.display.Stage;
  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.geom.Rectangle;

  public class PrintMultiplePages extends MovieClip
  {
    private var sheet1:Sprite;
    private var sheet2:Sprite;

    public function PrintMultiplePages():void
    {
      init();
      printPages();
    }

    private function init():void
    {
      sheet1 = new Sprite();
      createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80,
  height:130});
      sheet2 = new Sprite();
```

```
    createSheet(sheet2, "There was a great story to tell, and it ended
quickly.\n\nThe end.", null);
  }

  private function createSheet(sheet:Sprite, str:String,
imgValue:Object):void
  {
    sheet.graphics.beginFill(0xEEEEEE);
    sheet.graphics.lineStyle(1, 0x000000);
    sheet.graphics.drawRect(0, 0, 100, 200);
    sheet.graphics.endFill();

    var txt:TextField = new TextField();
    txt.height = 200;
    txt.width = 100;
    txt.wordWrap = true;
    txt.text = str;

    if (imgValue != null)
    {
      var img:Sprite = new Sprite();
      img.graphics.beginFill(0xFFFFFF);
      img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width,
imgValue.height);
      img.graphics.endFill();
      sheet.addChild(img);
    }
    sheet.addChild(txt);
  }

  private function printPages():void
  {
    var pj:PrintJob = new PrintJob();
    var pagesToPrint:uint = 0;
    if (pj.start())
    {
      if (pj.orientation == PrintJobOrientation.LANDSCAPE)
      {
        throw new Error("Page is not set to an orientation of
portrait.");
      }

      sheet1.height = pj.pageHeight;
      sheet1.width = pj.pageWidth;
      sheet2.height = pj.pageHeight;
      sheet2.width = pj.pageWidth;

      try
      {
        pj.addPage(sheet1);
```

```
            pagesToPrint++;
          }
          catch (error:Error)
          {
          // 오류에 응답합니다 .
          }

          try
          {
            pj.addPage(sheet2);
            pagesToPrint++;
          }
          catch (error:Error)
          {
            // 오류에 응답합니다 .
          }

          if (pagesToPrint > 0)
          {
            pj.send();
          }
        }
      }
    }
}
```

예제: 배율 조절, 자르기 및 자동 맞춤

경우에 따라 화면상에 표시되는 방식과 용지에 인쇄되는 방식의 차이에 따라 인쇄할 때 표시 객체의 크기나 다른 속성을 조절할 수 있습니다. `scaleX` 및 `scaleY` 속성을 사용하는 등 인쇄하기 전에 표시 객체의 속성을 조절할 때 객체 크기가 인쇄 영역에 정의된 사각형보다 크면 객체가 잘립니다. 페이지를 인쇄한 후에는 속성을 원래대로 설정할 수도 있습니다.

다음 코드에서는 `txt` 표시 객체(녹색 상자 배경 제외)의 크기를 조절하고 지정된 사각형의 크기에 따라 텍스트 필드가 잘립니다. 인쇄한 후에는 텍스트 필드가 화면상에 표시되는 원래 크기로 돌아갑니다. 사용자가 운영 체제의 [인쇄] 대화 상자에서 인쇄 작업을 취소하면 Flash Player의 내용이 변경되어 작업을 취소한 사용자에게 경고를 표시합니다.

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.display.Stage;
  import flash.geom.Rectangle;

  public class PrintScaleExample extends Sprite
  {
```

```
private var bg:Sprite;
private var txt:TextField;

public function PrintScaleExample():void
{
  init();
  draw();
  printPage();
}

private function printPage():void
{
  var pj:PrintJob = new PrintJob();
  txt.scaleX = 3;
  txt.scaleY = 2;
  if (pj.start())
  {
    trace(">> pj.orientation: " + pj.orientation);
    trace(">> pj.pageWidth: " + pj.pageWidth);
    trace(">> pj.pageHeight: " + pj.pageHeight);
    trace(">> pj.paperWidth: " + pj.paperWidth);
    trace(">> pj.paperHeight: " + pj.paperHeight);

    try
    {
      pj.addPage(this, new Rectangle(0, 0, 100, 100));
    }
    catch (error:Error)
    {
      // 아무 작업도 수행하지 않습니다 .
    }
    pj.send();
  }
  else
  {
    txt.text = "Print job canceled";
  }
  // txt 배율 속성을 재설정합니다 .
  txt.scaleX = 1;
  txt.scaleY = 1;
}

private function init():void
{
  bg = new Sprite();
  bg.graphics.beginFill(0x00FF00);
  bg.graphics.drawRect(0, 0, 100, 200);
  bg.graphics.endFill();

  txt = new TextField();
```

```
        txt.border = true;
        txt.text = "Hello World";
      }

      private function draw():void
      {
        addChild(bg);
        addChild(txt);
        txt.x = 50;
        txt.y = 50;
      }
    }
}
```

제 25 장

External API 사용

25

ActionScript 3.0 External API를 사용하면 Adobe Flash Player 9가 실행되는 컨테이너 응용 프로그램과 ActionScript 간에 간단하게 통신할 수 있습니다. External API를 사용해야 하는 경우는 여러 가지가 있습니다. HTML 페이지에서 SWF 문서와 JavaScript 간에 상호 작용을 만들거나 Flash Player를 사용하여 SWF 파일을 표시하는 데스크톱 응용 프로그램을 작성하는 경우를 예로 들 수 있습니다.

이 장에서는 External API를 사용하여 컨테이너 응용 프로그램과 상호 작용하는 방법, HTML 페이지의 JavaScript와 ActionScript 간에 데이터를 전달하는 방법 및 ActionScript와 데스크톱 응용 프로그램 간에 통신을 구축하고 데이터를 교환하는 방법에 대해 설명합니다.

참고	이 장에서는 SWF의 ActionScript와 해당 SWF가 로드된 Flash Player 인스턴스를 참조하는 컨테이너 응용 프로그램 간의 통신에 대해서만 다룹니다. 응용 프로그램 내에서 Flash Player를 사용하는 방법은 여기서 설명하지 않습니다. Flash Player는 브라우저 플러그인 또는 프로젝터(독립 실행형 응용 프로그램)용으로 만들어졌습니다. 그 밖의 사용은 지원이 제한될 수 있습니다.

목차

External API 사용의 기초

External API 사용 소개

경우에 따라 자체적으로 SWF 파일을 실행할 수도 있지만(예: SWF 프로젝터를 만든 경우), 대부분의 경우 SWF 응용 프로그램은 다른 응용 프로그램 내부의 요소로 실행됩니다. SWF를 포함하는 컨테이너는 대개 HTML 파일입니다. 그러나 데스크톱 응용 프로그램의 사용자 인터페이스 전체 또는 일부에 SWF 파일을 사용하기도 합니다.

고급 기능을 갖춘 응용 프로그램을 사용하다 보면 SWF 파일과 컨테이너 응용 프로그램 간의 통신 설정이 필요할 수 있습니다. 예를 들어, 웹 페이지에서는 텍스트 또는 기타 정보를 HTML로 표시하고 SWF 파일을 포함시켜 차트나 비디오 등의 동적인 시각적 정보를 표시할 수 있습니다. 이 경우 사용자가 웹 페이지의 버튼을 클릭할 때 SWF 파일의 내용이 변경되도록 할 수 있습니다. ActionScript에는 SWF 파일의 ActionScript와 컨테이너 응용 프로그램의 기타 코드 사이에 이러한 통신을 가능하게 하는 External API라는 메커니즘이 포함되어 있습니다.

일반적인 External API 작업

이 장에서는 다음과 같은 일반적인 External API 작업에 대해 설명합니다.

- 컨테이너 응용 프로그램에 대한 정보 얻기
- ActionScript를 사용하여 웹 페이지 또는 데스크톱 응용 프로그램과 같은 컨테이너 응용 프로그램의 코드 호출
- 컨테이너 응용 프로그램의 코드에서 ActionScript 코드 호출
- 프록시를 통한 컨테이너 응용 프로그램의 ActionScript 코드 호출 간소화

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- ActiveX 컨테이너: 응용 프로그램 내에 SWF 내용을 표시해 주는 Flash Player ActiveX 컨트롤의 인스턴스가 포함된 컨테이너 응용 프로그램입니다(웹 브라우저 아님).
- 컨테이너 응용 프로그램: Flash Player가 SWF 파일을 실행하는 응용 프로그램으로, Flash Player 내용이 포함된 웹 브라우저 및 HTML 페이지 등이 있습니다.
- 프로젝터: SWF 파일의 내용과 Flash Player를 포함하는 독립 실행형 실행 파일로 변환된 SWF 파일입니다. 프로젝터는 Adobe Flash CS3 Professional이나 독립 실행형 Flash Player를 사용하여 만들 수 있습니다. 프로젝터는 CD-ROM을 통해 SWF 파일을 배포하거나, 다운로드 크기가 문제되지 않고 SWF 제작자가 Flash Player의 설치 여부와 관계없이 사용자가 SWF 파일을 실행할 수 있도록 하려는 유사 환경에서 SWF 파일을 배포하는 데 일반적으로 사용됩니다.

- 프록시: 특정 응용 프로그램("외부 응용 프로그램")에서 다른 응용 프로그램("호출 응용 프로그램") 대신 코드를 호출하여 호출 응용 프로그램으로 값을 반환하는 중개 응용 프로그램 또는 코드입니다. 프록시는 다음과 같은 여러 가지 용도에 사용될 수 있습니다.
  - 호출 응용 프로그램의 네이티브 함수 호출을 외부 응용 프로그램이 이해할 수 있는 형식으로 변환하여 외부 함수 호출 프로세스 간소화
  - 호출자가 외부 응용 프로그램과 직접 통신하는 것을 막는 보안 또는 기타 제한 사항 문제 해결
- 직렬화: 객체 또는 데이터 값을 인터넷 환경이나 단일 컴퓨터에서 실행되는 두 개의 응용 프로그램과 같은 두 개 프로그래밍 시스템 간 메시지를 통한 값 전달에 사용할 수 있는 형식으로 변환하는 작업입니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플 중 상당수는 전체 작업 예제 또는 값 확인용 코드가 아니라 데모용 소형 코드 목록입니다. External API를 사용하려면 컨테이너 응용 프로그램 코드와 함께 ActionScript 코드를 작성해야 하므로, 컨테이너(예: SWF가 포함된 웹 페이지)를 만들고 코드 샘플을 사용하여 해당 컨테이너와 상호 작용하는 과정이 예제 테스트에 포함됩니다.

ActionScript와 JavaScript 간의 통신 예제를 테스트하려면:

1. 새 Flash 문서를 만들고 컴퓨터에 저장합니다.
2. 주 메뉴에서 [파일] > [제작 설정]을 선택합니다.
3. [제작 설정] 대화 상자의 [포맷] 탭에서 HTML 및 Flash 체크 상자가 선택되어 있는지 확인합니다.
4. [제작] 버튼을 클릭합니다. 이렇게 하면 Flash 문서 저장 시 사용한 것과 같은 이름의 SWF 파일과 HTML 파일이 같은 폴더에 생성됩니다. [확인]을 클릭하여 [제작 설정] 대화 상자를 닫습니다.
5. HTML 체크 상자의 선택을 취소합니다. 이제 HTML 페이지가 생성되었으므로 이를 수정하여 원하는 JavaScript 코드를 추가할 수 있습니다. HTML 체크 상자의 선택을 취소하면 HTML 페이지를 수정하더라도 Flash에서 SWF 파일을 제작할 때 새 HTML 문서로 사용자의 변경 사항을 덮어쓰는 일이 없습니다.
6. [확인]을 클릭하여 [제작 설정] 대화 상자를 닫습니다.
7. HTML 또는 텍스트 편집기 응용 프로그램을 사용하여 SWF 제작 시 Flash에서 만든 HTML 파일을 엽니다. HTML 소스 코드에서 스크립트 요소를 추가한 다음 이를 예제 코드 샘플의 JavaScript 코드에 복사합니다.
8. HTML 파일을 저장하고 Flash로 돌아갑니다.
9. 타임라인의 프레임 1에서 키프레임을 선택하고 [액션] 패널을 엽니다.

10. [스크립트] 창에 ActionScript 코드 샘플을 복사합니다.
11. 주 메뉴에서 [파일] > [제작]을 선택하여 SWF 파일에 변경 사항을 업데이트합니다.
12. 웹 브라우저에서 편집한 HTML 페이지를 열어 확인한 후 ActionScript와 HTML 간의 통신을 테스트합니다.

ActionScript와 ActiveX 컨테이너 간의 통신 예제를 테스트하려면:

1. 새 Flash 문서를 만들고 컴퓨터에 저장합니다. 해당 문서를 컨테이너 응용 프로그램의 SWF 파일 폴더에 저장하려면
2. 주 메뉴에서 [파일] > [제작 설정]을 선택합니다.
3. [제작 설정] 대화 상자의 [포맷] 탭에서 Flash 체크 상자만 선택되어 있는지 확인합니다.
4. Flash 체크 상자 옆에 있는 [파일] 필드에서 폴더 아이콘을 클릭하여 SWF 파일을 제작할 폴더를 선택합니다. SWF 파일 위치를 설정하여 Flash 문서를 한 폴더에 저장할 수도 있지만, 제작된 SWF 파일은 컨테이너 응용 프로그램의 소스 코드가 들어 있는 폴더 등 다른 폴더에 저장하는 것이 좋습니다.
5. 타임라인의 프레임 1에서 키프레임을 선택하고 [액션] 패널을 엽니다.
6. 예제의 ActionScript 코드를 [스크립트] 창으로 복사합니다.
7. 주 메뉴에서 [파일] > [제작]을 선택하여 SWF 파일을 다시 제작합니다.
8. 컨테이너 응용 프로그램을 만들고 실행하여 ActionScript와 컨테이너 응용 프로그램 간의 통신을 테스트합니다.

이 장의 끝부분에 있는 두 가지 예제는 External API를 사용하여 HTML 페이지 및 C# 데스크톱 응용 프로그램과 각각 통신하는 전체 예제입니다. 이들 예제에는 ActionScript 및 컨테이너 오류 검사 코드를 비롯한 전체 코드가 포함되어 있으며, External API로 코드를 작성할 때 이 코드를 사용해야 합니다. External API를 사용하는 기타 전체 예제는 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 ExternalInterface 클래스에 대한 클래스 예제를 참조하십시오.

External API 요구 사항 및 장점

External API는 ActionScript의 일부로서, Flash Player의 컨테이너 역할을 하는 "외부 응용 프로그램"(일반적으로 웹 브라우저 또는 독립 실행형 프로젝터 응용 프로그램)에서 실행되는 코드와 ActionScript 간의 통신 메커니즘을 제공합니다. ActionScript 3.0에서는 ExternalInterface 클래스에서 External API의 기능을 제공합니다. Flash Player 8의 이전 버전에서는 `fscommand()` 액션을 사용하여 컨테이너 응용 프로그램과 통신합니다. Flash Player 8 버전부터는 ExternalInterface 클래스가 `fscommand()`를 대체하므로, JavaScript와 ActionScript 간의 모든 통신에 이 클래스를 사용하는 것이 좋습니다.

예외	예를 들어 이전 응용 프로그램과 호환성을 유지해야 하거나 타사 SWF 컨테이너 응용 프로그램 또는 독립 실행형 Flash Player와의 호환을 위해 이전 `fscommand()` 함수를 사용해야 하는 경우, flash.system 패키지에서 패키지 레벨 함수로 사용할 수 있습니다.

ExternalInterface 클래스는 ActionScript 및 Flash Player가 HTML 페이지의 JavaScript 또는 Flash Player 인스턴스를 포함하는 데스크톱 응용 프로그램과 쉽게 통신할 수 있도록 하는 하위 시스템입니다.

ExternalInterface 클래스는 다음 경우에만 사용할 수 있습니다.

- 지원되는 모든 버전의 Internet Explorer for Windows(5.0 이상)
- Flash Player ActiveX 컨트롤의 인스턴스를 사용하는 데스크톱 응용 프로그램 등 컨테이너 응용 프로그램
- NPRuntime 인터페이스를 지원하는 브라우저(현재 다음과 같은 브라우저가 있습니다.)
  - Firefox 1.0 이상
  - Mozilla 1.7.5 이상
  - Netscape 8.0 이상
  - Safari 1.3 이상

그 밖의 모든 경우(예: 독립 실행형 플레이어에서 실행) `ExternalInterface.available` 속성은 `false`를 반환합니다.

ActionScript에서는 HTML 페이지에서 JavaScript 함수를 호출할 수 있습니다. External API는 `fscommand()`와 비교할 때 다음과 같은 향상된 기능을 제공합니다.

- `fscommand()` 함수와 함께 사용되는 함수는 물론 모든 JavaScript 함수를 사용할 수 있습니다.
- 이름과 개수에 상관없이 인수를 전달할 수 있습니다. 명령과 단일 문자열 인수를 전달하는 데 제한이 없습니다. 따라서 External API는 `fscommand()`보다 유연성이 뛰어납니다.
- Boolean, Number 및 String 등 다양한 유형의 데이터를 전달할 수 있습니다. 더 이상 String 매개 변수에 국한되지 않습니다.

- 호출 값을 수신할 수 있으며 그 값은 즉시 사용자 호출의 반환 값으로 ActionScript에 반환됩니다.

경고
HTML 페이지의 Flash Player 인스턴스에 제공된 이름(`object` 태그의 `id` 속성)에 JavaScript에서 연산자로 정의된 하이픈(`-`)이나 다른 문자가 포함되어 있으면 (예: `+`, `*`, `/`, `\`, `.` 등) Internet Explorer에서 컨테이너 웹 페이지를 보면서 ActionScript에서 ExternalInterface를 호출할 수 없습니다. 뿐만 아니라 Flash Player 인스턴스를 정의하는 HTML 태그(`object` 및 `embed` 태그)가 HTML `form` 태그에 중첩되어 있는 경우에도 ActionScript에서 ExternalInterface가 호출되지 않습니다.

ExternalInterface 클래스 사용

ActionScript와 컨테이너 응용 프로그램 간의 통신은 다음 두 가지 양식 중 하나를 사용할 수 있습니다. 즉, ActionScript가 컨테이너에 정의된 코드(예: JavaScript 함수)를 호출하거나 컨테이너의 코드가 호출 가능한 것으로 지정된 ActionScript 함수를 호출할 수 있습니다. 두 경우에 호출되는 코드에 정보를 전송하고 호출을 만드는 코드에 결과를 반환할 수 있습니다.

ExternalInterface 클래스에는 이러한 통신을 쉽게 하기 위한 두 개의 정적 속성과 두 개의 정적 메서드가 포함되어 있습니다. 이러한 속성과 메서드를 사용하여 외부 인터페이스 연결에 대한 정보를 가져오고, ActionScript에서 컨테이너의 코드를 실행하고, 컨테이너에서 ActionScript 함수를 호출할 수 있는 상태로 만듭니다.

외부 컨테이너에 대한 정보 얻기

`ExternalInterface.available` 속성은 현재 Flash Player가 외부 인터페이스를 제공하는 컨테이너에 있는지 여부를 나타냅니다. 외부 인터페이스를 사용할 수 있는 경우 이 속성은 `true`이고, 그렇지 않으면 `false`입니다. ExternalInterface 클래스에서 다른 함수를 사용하기 전에 항상 다음과 같이 현재 컨테이너에서 외부 인터페이스 통신을 지원하는지 확인해야 합니다.

```
if (ExternalInterface.available)
{
  // 여기에서 ExternalInterface 메서드 호출을 수행합니다.
}
```

참조
`ExternalInterface.available` 속성은 현재 컨테이너가 ExternalInterface 연결을 지원하는 유형인지 여부를 보고합니다. 현재 브라우저에서 JavaScript가 활성화되어 있는지 여부는 표시되지 않습니다.

ExternalInterface.objectID 속성을 사용하면 Flash Player 인스턴스의 고유한 ID(특히 Internet Explorer에서 object 태그의 id 특성 또는 NPRuntime 인터페이스를 사용하는 브라우저에서 embed 태그의 name 특성)를 확인할 수 있습니다. 이 고유한 ID는 브라우저에 있는 현재 SWF 문서를 나타내고 SWF 문서에 대한 참조를 만드는 데 사용할 수 있습니다. 예를 들어, 컨테이너 HTML 페이지에서 JavaScript 함수를 호출할 때 사용할 수 있습니다. Flash Player 컨테이너가 웹 브라우저가 아니면 이 속성은 null입니다.

ActionScript에서 외부 코드 호출

ExternalInterface.call() 메서드는 컨테이너 응용 프로그램에서 코드를 실행합니다. 이때 컨테이너 응용 프로그램에서 호출할 함수 이름이 포함된 문자열이 매개 변수로 반드시 필요합니다. ExternalInterface.call() 메서드에 전달된 그 밖의 매개 변수는 함수 호출의 매개 변수로 컨테이너에 전달됩니다.

```
// 외부 함수 "addNumbers"를 호출합니다.
// 두 개의 매개 변수를 전달하고 변수 "result"에 해당 함수의 결과를
// 지정합니다.
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

컨테이너가 HTML 페이지이면 이 메서드는 해당 HTML 페이지의 script 요소에 정의된 이름으로 JavaScript 함수를 호출합니다. JavaScript 함수의 반환 값은 ActionScript로 다시 전달됩니다.

```
<script language="JavaScript">
  // 두 숫자를 더하고 결과를 ActionScript에 다시 보냅니다.
  function addNumbers(num1, num2)
  {
    return (num1 + num2);
  }
</script>
```

컨테이너가 다른 ActiveX 컨테이너인 경우 이 메서드로 인해 Flash Player ActiveX 컨트롤이 해당 FlashCall 이벤트를 전달합니다. 지정된 함수 이름 및 매개 변수는 Flash Player에서 XML 문자열로 직렬화합니다. 컨테이너는 event 객체의 request 속성에서 이 정보에 액세스할 수 있으며 이 정보를 통해 자신의 코드를 어떻게 실행할지 결정합니다. 컨테이너 코드는 ActionScript에 값을 반환하기 위해 ActiveX 객체의 SetReturnValue() 메서드를 호출하여 그 결과(XML 문자열로 직렬화된 결과)를 해당 메서드의 매개 변수로 전달합니다. 이 통신에 사용되는 XML 형식에 대한 자세한 내용은 691페이지의 “External API의 XML 형식”을 참조하십시오.

컨테이너가 웹 브라우저인지 또는 다른 ActiveX 컨테이너인지 관계없이 호출이 실패하거나 컨테이너 메서드에서 반환 값을 지정하지 않으면 null이 반환됩니다. 포함 환경이 호출 코드가 액세스할 수 없는 보안 샌드박스에 속하는 경우 ExternalInterface.call() 메서드가 SecurityError 예외를 발생시킵니다. 컨테이너 환경에서 allowScriptAccess에 적절한 값을 설정하면 이 문제를 해결할 수 있습니다. 예를 들어 HTML 페이지에서 allowScriptAccess의 값을 변경하려면 object 및 embed 태그에서 해당 특성을 편집합니다.

컨테이너에서 ActionScript 코드 호출

컨테이너는 함수에 있는 ActionScript 코드만 호출할 수 있고 다른 ActionScript 코드는 호출할 수 없습니다. 컨테이너 응용 프로그램에서 ActionScript 함수를 호출하려면 ExternalInterface 클래스로 함수를 등록한 다음 컨테이너의 코드에서 호출해야 합니다.

먼저 ActionScript 함수를 등록하여 컨테이너에서 사용할 수 있도록 나타내야 합니다. 다음과 같이 ExternalInterface.addCallback() 메서드를 사용합니다.

```
function callMe(name:String):String
{
  return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

addCallback() 메서드는 두 가지 매개 변수를 사용합니다. 먼저 문자열로 된 함수 이름에 의해 컨테이너에서 함수를 인식하게 됩니다. 두 번째 매개 변수는 컨테이너가 정의된 함수 이름을 호출할 때 실행될 실제 ActionScript 함수입니다. 이러한 이름은 고유하므로 실제 ActionScript 함수에 다른 이름이 있더라도 컨테이너에서 사용할 함수 이름을 지정할 수 있습니다. 이것은 함수 이름을 모르는 경우, 예를 들어, 익명 함수가 지정되거나 호출할 함수가 런타임에 확인되는 경우에 특히 유용합니다.

ActionScript 함수는 ExternalInterface 클래스로 등록되었으므로 컨테이너에서 실제로 이 함수를 호출할 수 있습니다. 실행 방법은 컨테이너 유형에 따라 다양합니다. 예를 들어 웹 브라우저의 JavaScript 코드에서 ActionScript 함수는 Flash Player 브라우저 객체의 메서드(object 또는 embed 태그를 나타내는 JavaScript 객체의 메서드)와 마찬가지로 등록된 함수 이름을 사용하여 호출됩니다. 즉, 로컬 함수가 호출되는 것처럼 매개 변수가 전달되고 결과가 반환됩니다.

```
<script language="JavaScript">
  // callResult가 값 "busy signal"을 가져옵니다.
  var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
  ...
  <embed name="flashObject".../>
</object>
```

또는 데스크톱 응용 프로그램에서 실행되는 SWF 파일에서 ActionScript 함수를 호출할 때 등록된 함수 이름 및 매개 변수를 XML 형식 문자열로 직렬화해야 합니다. 그러면 XML 문자열을 매개 변수로 ActiveX 컨트롤의 `CallFunction()` 메서드를 호출하는 방식으로 호출이 실제로 수행됩니다. 이 통신에 사용되는 XML 형식에 대한 자세한 내용은 691페이지의 “External API의 XML 형식”을 참조하십시오.

두 경우 모두 ActionScript 함수의 반환 값은 다시 컨테이너 코드로 전달됩니다. 호출자가 브라우저의 JavaScript인 경우에는 직접 값으로 전달되고 호출자가 ActiveX 컨테이너인 경우에는 XML 형식 문자열로 직렬화되어 전달됩니다.

External API의 XML 형식

Shockwave Flash ActiveX 컨트롤을 호스팅하는 ActionScript와 응용 프로그램 간의 통신은 특정 XML 형식을 사용하여 함수 호출과 값을 인코딩합니다. External API에서 사용하는 XML 형식에는 두 부분이 있습니다. 한 가지 형식은 함수 호출을 나타내는 데 사용됩니다. 다른 형식은 개별 값을 나타내는 데 사용됩니다. 이 형식은 함수의 매개 변수와 함수 반환 값에 사용됩니다. 함수 호출의 XML 형식은 ActionScript에서 송수신되는 호출에 사용됩니다.

ActionScript에서 함수를 호출하는 경우에는 Flash Player가 XML을 컨테이너에 전달하고, 컨테이너에서 호출하는 경우에는 Flash Player에서 컨테이너 응용 프로그램이 이 형식으로 XML 문자열을 전달해야 합니다. 다음 XML 프래그먼트는 XML 형식의 함수 호출을 보여 줍니다.

```
<invoke name="functionName" returntype="xml">
  <arguments>
    ... (individual argument values)
  </arguments>
</invoke>
```

루트 노드는 `invoke` 노드입니다. 이 노드에는 호출할 함수 이름을 나타내는 `name`과 항상 `xml`인 `returntype`의 두 가지 특성이 있습니다. 함수 호출에 매개 변수가 포함되는 경우 `invoke` 노드에 자식 `arguments` 노드가 포함되고 그 노드의 자식 노드는 아래 설명하는 개별 값 형식의 매개 변수 값이 됩니다.

함수 매개 변수와 함수 반환 값 등의 개별 값은 실제 값뿐 아니라 데이터 형식 정보가 포함된 형식 지정 스키마를 사용합니다. 다음 표에는 ActionScript 클래스 및 해당 데이터 유형의 값을 인코딩하는 데 사용되는 XML 형식이 나와 있습니다.

ActionScript 클래스/값	C# 클래스/값	형식	주석
null	null	`<null/>`	
Boolean `true`	bool true	`<true/>`	
Boolean `false`	bool false	`<false/>`	
String	string	`<string>문자열 값</string>`	
Number, int, uint	single, double, int, uint	`<number>27.5</number>` `<number>-12</number>`	

Array(요소는 혼합 유형일 수 있습니다.)	ArrayList 또는 object[] 등 혼합 유형 요소를 허용하는 컬렉션	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			`property` 노드는 개별 요소를 정의하고 `id` 특성은 숫자 0부터 시작하는 인덱스입니다.

Object	문자열 키가 있는 HashTable처럼 문자열 키와 객체 값이 있는 사전	`<object>`
		`<property id="이름">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="나이">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			`property` 노드는 개별 속성을 정의하고 `id` 특성은 속성 이름(문자열)입니다.

기타 기본 제공 또는 사용자 정의 클래스		`<null/>`
		또는
		`<object></object>`
			ActionScript에서는 다른 객체를 null 또는 빈 객체로 인코딩합니다. 두 경우 모두 속성 값이 사라집니다.

참고 이 표에서는 예제로 ActionScript 클래스와 함께 해당 C# 클래스를 보여 줍니다. 그러나 External API는 ActiveX 컨트롤을 지원하는 모든 프로그래밍 언어 또는 런타임 통신에 사용될 수 있으며 C# 응용 프로그램에만 사용하는 것이 아닙니다.

External API와 ActiveX 컨테이너 응용 프로그램을 사용하여 직접 응용 프로그램을 작성할 때는 네이티브 함수 호출을 직렬화된 XML 형식으로 변환하는 작업을 수행할 프록시를 작성하는 것이 편하다는 것을 알게 될 것입니다. C#으로 작성된 proxy 클래스 예제는 704페이지의 “ExternalInterfaceProxy 클래스 내부”를 참조하십시오.

예제: 웹 페이지 컨테이너에서 External API 사용

이 샘플 응용 프로그램은 다른 사람과 채팅이 가능한 인스턴트 메시징 응용 프로그램(응용 프로그램 이름은 Introvert IM)을 사용할 때 웹 브라우저의 ActionScript와 JavaScript 간의 적절한 통신 기법을 보여 줍니다. External API를 사용하여 웹 페이지와 SWF 인터페이스의 HTML 양식 간에 메시지가 전송됩니다. 이 예제에서 보여 주는 기술은 다음과 같습니다.

- 통신을 설정하기 전에 브라우저가 통신 가능한 상태인지 확인하여 제대로 통신 초기화
- 컨테이너의 External API 지원 여부 확인
- ActionScript에서 JavaScript 함수 호출, 매개 변수 전달 및 응답으로 값 수신
- ActionScript 메서드를 JavaScript에서 호출할 수 있는 상태로 만들고 이러한 호출 수행

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. Introvert IM 응용 프로그램 파일은 Samples/IntrovertIM_HTML 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
IntrovertIMApp.fla 또는 IntrovertIMApp.mxml	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/introvertIM/IMManager.as	ActionScript 및 컨테이너 간에 통신을 구축하고 관리하는 클래스입니다.
com/example/programmingas3/introvertIM/IMMessageEvent.as	컨테이너에서 메시지를 수신할 때 IMManager 클래스에서 전달되는 사용자 정의 이벤트 유형입니다.

파일	설명
com/example/programmingas3/introvertIM/IMStatus.as	값이 응용 프로그램에서 선택할 수 있는 다른 "가용성" 상태 값을 나타내는 열거입니다.
html-flash/IntrovertIMApp.html 또는 html-template/index.template.html	Flash 응용 프로그램의 HTML 페이지(html-flash/IntrovertIMApp.html)이거나 Adobe Flex 응용 프로그램의 컨테이너 HTML 페이지(html-template/index.template.html) 작성에 사용된 템플릿입니다. 이 파일에는 응용 프로그램의 컨테이너 부분을 구성하는 모든 JavaScript 함수가 포함되어 있습니다.

ActionScript 브라우저 통신 준비

External API는 주로 ActionScript 응용 프로그램이 웹 브라우저와 통신할 수 있도록 허용하는 데 사용됩니다. External API를 사용하면 ActionScript 메서드가 JavaScript에서 작성한 코드를 호출할 수 있으며 그 반대도 마찬가지입니다. 브라우저의 복잡성과 내부에서 페이지를 렌더링하는 방법으로 인해 HTML 페이지의 첫 번째 JavaScript를 실행하기 전에 SWF 문서에서 콜백을 등록하도록 보장할 수 있는 방법은 없습니다. 이러한 이유로 JavaScript에서 SWF 문서의 함수를 호출하기 전에 항상 SWF 문서에서 HTML 페이지를 호출하여 SWF 문서가 연결을 허용할 수 있는 상태임을 알려야 합니다.

예를 들어, IMManager 클래스에서 수행한 몇 가지 단계를 통해 Introvert IM에서 브라우저가 통신할 수 있는 상태인지 확인하고 통신을 위해 SWF 파일을 준비합니다. 브라우저가 통신 가능한 상태인지 확인하는 첫 번째 단계는 다음과 같이 IMManager 생성자에서 이루어집니다.

```
public function IMManager(initialStatus:IMStatus)
{
  _status = initialStatus;

  // 컨테이너에서 외부 API 를 사용할 수 있는지 확인합니다 .
  if (ExternalInterface.available)
  {
    try
    {
      // isContainerReady() 메서드가 호출되고 , 이 메서드는 다시
      // 컨테이너를 호출하여 Flash Player 가 로드되었으며 컨테이너에서
      // SWF 의 호출을 수신할 준비가 되었는지 확인합니다 .
      var containerReady:Boolean = isContainerReady();
      if (containerReady)
      {
        // 컨테이너가 준비되었으면 SWF 의 함수를 등록합니다 .
        setupCallbacks();
      }
```

```
        else
        {
          // 컨테이너가 준비되지 않았으면 100ms 간격으로 컨테이너를 호출하도록
          // Timer를 설정합니다. 준비가 되면 타이머가 중지됩니다.
          var readyTimer:Timer = new Timer(100);
          readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
          readyTimer.start();
        }
      }
      ...
    }
    else
    {
      trace("이 컨테이너에는 외부 인터페이스를 사용할 수 없습니다.");
    }
}
```

우선 이 코드는 `ExternalInterface.available` 속성을 사용하여 현재 컨테이너에서도 External API를 사용 가능한지 여부를 확인합니다. 가용성 상태인 경우 통신을 설정하는 프로세스가 시작됩니다. 외부 응용 프로그램과 통신을 시도하면 보안 예외 및 기타 오류가 발생할 수 있으므로 이 코드는 `try` 블록에서 래핑됩니다(편의상 해당 `catch` 블록은 샘플에서 생략).

그런 다음 이 코드는 여기에 나열된 `isContainerReady()` 메서드를 호출합니다.

```
private function isContainerReady():Boolean
{
  var result:Boolean = ExternalInterface.call("isReady");
  return result;
}
```

이제 `isContainerReady()` 메서드가 다시 `ExternalInterface.call()` 메서드를 사용하여 JavaScript 함수 `isReady()`를 호출합니다.

```
<script language="JavaScript">
<!--
// ------- 전용 변수 -------
var jsReady = false;
...
// ------- ActionScript에서 호출되는 함수 -------
// 페이지가 초기화되었고 JavaScript가 사용 가능한지 확인하기 위해 호출됩니다.
function isReady()
{
  return jsReady;
}
...
// <body> 태그의 onload 이벤트에 의해 호출됩니다.
function pageInit()
{
  // JavaScript가 준비되었음을 기록합니다.
  jsReady = true;
}
```

```
...
//-->
</script>
```

isReady() 함수는 jsReady 변수 값만 반환합니다. 처음에 이 변수는 false이며, 웹 페이지의 onload 이벤트가 트리거되면 변수 값이 true로 바뀝니다. 즉, 페이지가 로드되기 전에 ActionScript가 isReady() 함수를 호출하면 JavaScript가 ExternalInterface.call("isReady")에 false를 반환하고, 그 결과 ActionScript isContainerReady() 메서드가 false를 반환합니다. 페이지가 로드되면 JavaScript isReady() 함수가 true를 반환하므로 ActionScript isContainerReady() 메서드도 true를 반환합니다.

IMManager 생성자로 돌아가면 컨테이너의 준비 상태에 따라 두 가지 중 하나가 발생합니다. isContainerReady()가 true를 반환하면 코드가 setupCallbacks() 메서드만 호출하여 JavaScript와 통신을 설정하는 프로세스를 완료합니다. 그와 반대로 isContainerReady()가 false를 반환하면 프로세스는 기본적으로 보류됩니다. Timer 객체가 만들어지고 100밀리초마다 timerHandler() 메서드를 호출하도록 지시됩니다.

```
private function timerHandler(event:TimerEvent):void
{
  // 컨테이너가 이제 준비되었는지 확인합니다 .
  var isReady:Boolean = isContainerReady();
  if (isReady)
  {
    // 컨테이너가 준비되면 더 이상 확인할 사항이 없으므로
    // 타이머를 중지합니다 .
    Timer(event.target).stop();
    // 컨테이너에서 호출할 수 있는 ActionScript 메서드를
    // 설정합니다 .
    setupCallbacks();
  }
}
```

timerHandler() 메서드가 호출될 때마다 isContainerReady() 메서드의 결과를 다시 확인합니다. 컨테이너가 초기화되면 이 메서드는 true를 반환합니다. 그런 다음 이 코드는 Timer를 중지하고 setupCallbacks() 메서드를 호출하여 브라우저와의 통신 설정 프로세스를 마칩니다.

JavaScript에 ActionScript 메서드 표시

앞의 예제와 같이, 브라우저가 대기 상태임을 확인한 코드는 `setupCallbacks()` 메서드를 호출합니다. 이 메서드는 ActionScript를 준비하여 다음과 같이 JavaScript에서 호출을 수신합니다.

```
private function setupCallbacks():void
{
  // 컨테이너에 SWF 클라이언트 함수를 등록합니다 .
  ExternalInterface.addCallback("newMessage", newMessage);
  ExternalInterface.addCallback("getStatus", getStatus);
  // SWF 의 호출 준비가 완료되었음을 컨테이너에 알립니다 .
  ExternalInterface.call("setSWFIsReady");
}
```

`setCallBacks()` 메서드는 `ExternalInterface.addCallback()`을 호출하여 JavaScript에서 호출 가능한 두 개의 메서드를 등록함으로써 컨테이너와 통신할 준비를 마칩니다. 이 코드의 첫 번째 매개 변수는 ActionScript의 메서드 이름과 같습니다. 이 메서드는 JavaScript에 이 이름으로 인식됩니다(`"newMessage"` 및 `"getStatus"`). (이 경우 다른 이름을 사용해도 별 다른 장점이 없으므로 간편하게 같은 이름을 다시 사용합니다.) 마지막으로 `ExternalInterface.call()` 메서드를 사용하여 JavaScript 함수 `setSWFIsReady()`를 호출합니다. 이 함수는 컨테이너에 ActionScript 함수가 등록되었음을 알려 줍니다.

ActionScript에서 브라우저로 통신

Introvert IM 응용 프로그램은 컨테이너 페이지에서 JavaScript 함수를 호출하는 다양한 예를 보여 줍니다. 가장 간단한 경우(`setupCallbacks()` 메서드의 예제)는 매개 변수를 전달하거나 반환값을 수신하지 않고 JavaScript 함수 `setSWFIsReady()`를 호출하는 것입니다.

```
ExternalInterface.call("setSWFIsReady");
```

`isContainerReady()`의 다른 예제에서 ActionScript는 `isReady()` 함수를 호출하고 그 응답으로 부울 값을 수신합니다.

```
var result:Boolean = ExternalInterface.call("isReady");
```

External API를 사용하여 매개 변수를 JavaScript 함수로 전달할 수도 있습니다. 예를 들어 IMManager 클래스의 `sendMessage()` 메서드를 생각해 보십시오. 이 메서드는 사용자가 "대화 상자"에게 새 메시지를 보낼 때 호출됩니다.

```
public function sendMessage(message:String):void
{
  ExternalInterface.call("newMessage", message);
}
```

여기에서도 지정된 JavaScript 함수를 호출하고 브라우저에 새 메시지를 알리는 데 ExternalInterface.call()이 사용됩니다. 뿐만 아니라 메시지 자체가 추가 매개 변수로 ExternalInterface.call()에 전달되고, 그 결과 JavaScript 함수 newMessage()에 매개 변수로 전달됩니다.

JavaScript에서 ActionScript 코드 호출

통신은 두 갈래 길이라고 할 수 있으며 Introvert IM 응용 프로그램도 예외는 아닙니다. Flash Player IM 클라이언트만 JavaScript를 호출하여 메시지를 보내는 게 아니라 HTML 양식도 JavaScript 코드를 호출하여 SWF 파일에 메시지를 보내고 SWF 파일에서도 정보를 요구합니다. 예를 들어 SWF 파일이 컨테이너에 연락처 구축이 끝나 통신할 수 있는 상태가 되었음을 알리면 브라우저에서 처음 하는 일은 IMManager 클래스의 getStatus() 메서드를 호출하여 SWF IM 클라이언트에서 첫 사용자의 가용성 상태를 검색하는 것입니다. 웹 페이지에서 updateStatus() 함수로 다음과 같이 수행하면 됩니다.

```
<script language="JavaScript">
...
function updateStatus()
{
  if (swfReady)
  {
    var currentStatus = getSWF("IntrovertIMApp").getStatus();
    document.forms["imForm"].status.value = currentStatus;
  }
}
...
</script>
```

이 코드는 swfReady 변수의 값을 확인합니다. 이 값은 SWF 파일이 ExternalInterface 클래스로 메서드를 등록했음을 브라우저에 알렸는지 여부를 추적합니다. SWF 파일이 통신을 수신할 수 있는 상태가 되면 다음 줄(var currentStatus = ...)이 IMManager 클래스에서 실제로 getStatus() 메서드를 호출합니다. 코드의 이 줄에서는 다음 세 가지가 발생합니다.

- getSWF() JavaScript 함수가 호출되어 SWF 파일을 나타내는 JavaScript 객체에 대한 참조를 반환합니다. getSWF()에 전달된 매개 변수는 HTML 페이지에 둘 이상의 SWF 파일이 있는 경우 브라우저 객체가 반환되는지 여부를 결정합니다. 이 매개 변수에 전달된 값은 SWF 파일을 포함하는 데 사용되는 object 태그의 id 특성 및 embed 태그의 name 특성과 일치해야 합니다.
- SWF 파일에 대한 참조를 사용하면 getStatus() 메서드가 SWF 객체의 메서드인 것처럼 호출됩니다. 이 경우 함수 이름 "getStatus"가 사용됩니다.
  ExternalInterface.addCallback()을 사용하여 ActionScript 함수를 등록할 때 이 이름을 사용합니다.

- `getStatus()` ActionScript 메서드는 값을 반환하고 이 값이 `currentStatus` 변수에 지정됩니다. 그 다음 이 변수는 `status` 텍스트 필드의 내용(`value` 속성)으로 지정됩니다.

`sendMessage()` JavaScript 함수는 ActionScript 함수에 매개 변수를 전달하는 방법을 보여 줍니다. (`sendMessage()`는 HTML 페이지에서 사용자가 [전송] 버튼을 누를 때 호출되는 함수입니다.)

```
<script language="JavaScript">
...
function sendMessage(message)
{
  if (swfReady)
  {
    ...
    getSWF("IntrovertIMApp").newMessage(message);
  }
}
...
</script>
```

`newMessage()` ActionScript 메서드에는 매개 변수가 하나 필요하므로 JavaScript 코드의 `newMessage()` 메서드를 호출할 때 JavaScript `message` 변수를 매개 변수로 사용하여 ActionScript에 전달합니다.

브라우저 유형 검색

브라우저마다 내용에 액세스하는 방법이 다르므로, 다음 예제의 `getSWF()` JavaScript 함수에서 보듯이 항상 JavaScript로 사용자의 브라우저가 무엇인지 검색하고 해당 브라우저의 구문에 따라 Window 객체나 Document 객체를 사용하여 무비에 액세스해야 합니다.

```
<script language="JavaScript">
...
function getSWF(movieName)
{
  if (navigator.appName.indexOf("Microsoft") != -1)
  {
    return window[movieName];
  }
  else
  {
    return document[movieName];
  }
}
...
</script>
```

스크립트에서 사용자의 브라우저 종류를 검색하지 않으면 HTML 컨테이너에서 SWF 파일을 재생할 때 예상치 못한 비헤이비어가 표시될 수도 있습니다.

예제: ActiveX 컨테이너에서 External API 사용

이 예제는 External API를 사용하여 ActionScript와 ActiveX 컨트롤을 사용하는 데스크톱 응용 프로그램 간에 통신하는 방법을 보여 줍니다. 이 예제에서는 ActionScript 코드 및 동일한 SWF 파일을 비롯한 Introvert IM 응용 프로그램을 다시 사용합니다. 따라서 ActionScript에서 External API를 사용하는 방법은 설명하지 않습니다. 앞의 예제와 비슷하므로 이해하는 데 도움이 될 것입니다.

이 예제에서는 데스크톱 응용 프로그램이 Microsoft Visual Studio .NET을 사용하여 C#로 작성됩니다. 여기에서는 ActiveX 컨트롤을 사용하여 External API로 작업하는 특정 기술을 중심으로 설명합니다. 이 예제에서는 다음 내용을 보여 줍니다.

- Flash Player ActiveX 컨트롤을 호스팅하는 데스크톱 응용 프로그램에서 ActionScript 함수 호출
- ActionScript에서 함수 호출을 검색하고 ActiveX 컨테이너에서 처리
- 프록시 클래스를 사용하여 Flash Player에서 ActiveX 컨테이너로 전송하는 메시지에 사용하는 직렬화된 XML 형식의 정보 숨기기

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. Introvert IM C# 파일은 Samples/IntrovertIM_CSharp 폴더에 있습니다. 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
AppForm.cs	C# Windows Forms 인터페이스가 있는 기본 응용 프로그램 파일입니다.
bin/Debug/IntrovertIMApp.swf	응용 프로그램에서 로드한 SWF 파일입니다.
ExternalInterfaceProxy/ExternalInterfaceProxy.cs	외부 인터페이스 통신을 위해 ActiveX 컨트롤 주위에서 래퍼 역할을 하는 클래스입니다. ActionScript에서 호출을 송수신하는 메커니즘을 제공합니다.
ExternalInterfaceProxy/ExternalInterfaceSerializer.cs	Flash Player의 XML 형식 메시지를 .NET 객체로 변환하는 작업을 수행하는 클래스입니다.
ExternalInterfaceProxy/ExternalInterfaceEventArgs.cs	이 파일은 ExternalInterfaceProxy 클래스가 ActionScript의 함수 호출 리스너를 알려 주기 위해 사용하는 사용자 정의 Delegate 클래스와 Event Arguments 클래스라는 두 가지 C# 유형(클래스)를 정의합니다.

파일	설명
ExternalInterfaceProxy/ ExternalInterfaceCall.cs	이 클래스는 함수 이름과 매개 변수의 속성을 사용하여 ActionScript의 함수 호출을 ActiveX 컨테이너로 나타내는 value 객체입니다.
bin/Debug/IntrovertIMApp.swf	응용 프로그램에서 로드한 SWF 파일입니다.
obj/ AxInterop.ShockwaveFlashObjects.dll, obj/Interop.ShockwaveFlashObjects.dll	Visual Studio .NET에서 만든 래퍼 어셈블리로서, 관리 대상 코드에서 Flash Player(Adobe ShockwaveÆ Flash) ActiveX 컨트롤에 액세스하는 데 필요합니다.

Introvert IM C# 응용 프로그램 개요

이 샘플 응용 프로그램은 서로 통신하는 두 가지 인스턴트 메시징 클라이언트 프로그램(SWF 파일에 포함된 프로그램과 Windows Forms로 작성된 프로그램)을 나타냅니다. 사용자 인터페이스에는 Shockwave Flash ActiveX 컨트롤의 인스턴스가 포함되며 이 인스턴스에서 ActionScript IM 클라이언트를 포함하는 SWF 파일이 로드됩니다. 이 인터페이스에는 메시지 입력 필드(`MessageText`), 클라이언트 간에 전송된 메시지 사본의 표시 필드(`Transcript`), SWF IM 클라이언트에 설정된 가용성 상태를 표시하는 세 번째 필드(`Status`) 등 Windows Forms IM 클라이언트를 구성하는 각종 텍스트 필드도 들어 있습니다.

Shockwave Flash ActiveX 컨트롤 포함

Windows Forms 응용 프로그램에 Shockwave Flash ActiveX 컨트롤을 포함하려면 먼저 Microsoft Visual Studio 도구 상자에 추가해야 합니다.

도구 상자에 컨트롤을 추가하려면:

1. Visual Studio Toolbox를 엽니다.
2. Visual Studio 2003에서는 Windows Forms 섹션을 마우스 오른쪽 버튼으로 클릭하고 Visual Studio 2005에서는 아무 섹션이나 마우스 오른쪽 버튼으로 클릭합니다. 컨텍스트 메뉴에서 Visual Studio 2003의 경우 [항목 추가/제거]를 선택하고 Visual Studio 2005의 경우 [항목 선택...]을 선택합니다.

   그러면 [도구 상자 사용자 지정(2003)/도구 상자 항목 선택(2005)] 대화 상자가 열립니다.
3. Flash Player ActiveX 컨트롤을 포함하여 컴퓨터에서 사용 가능한 모든 COM 구성 요소가 표시된 [COM 구성 요소] 탭을 선택합니다.
4. Shockwave Flash 객체로 스크롤하여 선택합니다.

   이 항목이 목록에 없으면 시스템에 Flash Player ActiveX 컨트롤이 설치되어 있는지 확인합니다.

ActionScript와 ActiveX 컨테이너의 통신에 대한 이해

ActiveX 컨테이너 응용 프로그램에서 External API를 사용하여 통신하면 한 가지 중요한 차이만 제외하고 웹 브라우저와 통신하는 것과 비슷하게 작동합니다. 앞에서 설명한 것처럼 개발자와 관련하여 말하자면, ActionScript가 웹 브라우저와 통신하는 경우 함수가 직접 호출됩니다. 플레이어와 브라우저 간에 함수 호출과 응답을 전달하기 위해 형식을 지정하는 방법은 표시되지 않습니다. 그러나 External API를 사용하여 ActiveX 컨테이너 응용 프로그램과 통신할 때 Flash Player에서는 메시지(함수 호출 및 반환 값)를 특정 XML 형식으로 응용 프로그램에 전송하고 컨테이너 응용 프로그램의 함수 호출과 반환 값이 같은 XML 형식을 사용하도록 합니다. ActiveX 컨테이너 응용 프로그램의 개발자는 적절한 형식의 함수 호출과 응답을 이해하고 만들 수 있는 코드를 작성해야 합니다.

Introvert IM C# 예제에는 메시지 형식을 지정할 수 없는 클래스 집합이 포함되어 있습니다. 그 대신 ActionScript 함수를 호출하고 ActionScript에서 함수 호출을 수신할 때 표준 데이터 형식을 사용할 수 있습니다. ExternalInterfaceProxy 클래스는 다른 도우미 클래스와 함께 이 기능을 제공하며 모든 .NET 프로젝트에서 다시 사용하여 External API 통신을 사용하도록 도울 수 있습니다.

기본 응용 프로그램 양식(AppForm.cs)에서 인용한 다음 코드 부분은 ExternalInterfaceProxy 클래스를 사용하여 이루어지는 간단한 상호 작용을 보여 줍니다.

```
public class AppForm : System.Windows.Forms.Form
{
  ...
  private ExternalInterfaceProxy proxy;
  ...
  public AppForm()
  {
    ...
    // 프록시에서 ActionScript로부터 호출을 수신할 때 알림을 받도록
    // 이 응용 프로그램을 등록합니다 .
    proxy = new ExternalInterfaceProxy(IntrovertIMApp);
    proxy.ExternalInterfaceCall += new
  ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
    ...
  }
  ...
```

이 응용 프로그램은 `proxy`라는 ExternalInterfaceProxy 인스턴스를 선언하고 만든 다음 사용자 인터페이스(`IntrovertIMApp`)에 있는 Shockwave Flash ActiveX 컨트롤에 대한 참조를 전달합니다. 그런 다음 이 코드는 proxy의 `ExternalInterfaceCall` 이벤트를 수신할 수 있도록 `proxy_ExternalInterfaceCall()` 메서드를 등록합니다. Flash Player에서 함수 호출이 오면 ExternalInterfaceProxy 클래스에서 이 이벤트를 전달합니다. 이 이벤트를 구독하는 것이 C# 코드가 ActionScript의 함수 호출을 수신하고 응답하는 방식입니다.

ActionScript에서 함수 호출이 오면 ExternalInterfaceProxy 인스턴스(proxy)에서 호출을 수신하고, XML 형식의 이 호출을 변환한 다음 proxy의 ExternalInterfaceCall 이벤트에 대한 리스너 객체에게 알립니다. AppForm 클래스의 경우 proxy_ExternalInterfaceCall() 메서드가 다음과 같이 해당 이벤트를 처리합니다.

```
/// <summary>
/// ActionScript ExternalInterface 호출 시 프록시에 의해 호출됨
/// SWF에 의해 만들어짐
/// </summary>
private object proxy_ExternalInterfaceCall(object sender,
ExternalInterfaceCallEventArgs e)
{
  switch (e.FunctionCall.FunctionName)
  {
    case "isReady":
      return isReady();
    case "setSWFIsReady":
      setSWFIsReady();
      return null;
    case "newMessage":
      newMessage((string)e.FunctionCall.Arguments[0]);
      return null;
    case "statusChange":
      statusChange();
      return null;
    default:
      return null;
  }
}
...
```

메서드가 ExternalInterfaceCallEventArgs 인스턴스를 받습니다(예제의 인스턴스 이름은 e). 그러면 이 객체가 ExternalInterfaceCall 클래스의 인스턴스인 FunctionCall 속성을 갖게 됩니다.

ExternalInterfaceCall 인스턴스는 두 가지 속성이 있는 간단한 value 객체입니다. FunctionName 속성에는 ActionScript ExternalInterface.Call() 문에 지정된 함수 이름이 포함되어 있습니다. ActionScript에 추가된 매개 변수는 ExternalInterfaceCall 객체의 Arguments 속성에 포함됩니다. 이 경우 이벤트를 처리하는 메서드는 트래픽 관리자처럼 작동하는 switch 문일 뿐입니다. FunctionName 속성의 값(e.FunctionCall.FunctionName)에 따라 호출할 AppForm 클래스의 메서드가 결정됩니다.

앞의 코드 샘플에서 switch 문의 분기는 일반적인 메서드 호출 시나리오를 보여 줍니다. 예를 들어, 모든 메서드는 ActionScript에 값을 반환하거나(예: isReady() 메서드 호출) 다른 메서드 호출에서와 같이 null을 반환해야 합니다. ActionScript에서 전달되는 매개 변수에 액세스하는 방법은 newMessage() 메서드 호출(Arguments 배열의 첫 번째 요소인 매개 변수 e.FunctionCall.Arguments[0]와 함께 전달)에 표시됩니다.

ExternalInterfaceProxy 클래스를 사용하여 C#에서 ActionScript 함수를 호출하면 ActionScript에서 함수 호출을 수신하는 것보다 간단합니다. ActionScript 함수를 호출하려면 다음과 같이 ExternalInterfaceProxy 인스턴스의 `Call()` 메서드를 사용합니다.

```
  /// <summary>
  /// "보내기" 버튼을 누르면 호출됩니다.
  /// MessageText 텍스트 필드가 매개 변수로 전달됩니다.
  /// </summary>
  /// <param name="message">보낼 메시지</param>
  private void sendMessage(string message)
  {
    if (swfReady)
    {
      ...
      // ActionScript에서 newMessage 함수를 호출합니다.
      proxy.Call("newMessage", message);
    }
  }
  ...
  /// <summary>
  /// ActionScript 함수를 호출하여 현재 "가용성" 상태를 검색해서 텍스트
  /// 필드에 씁니다.
  /// </summary>
  private void updateStatus()
  {
    Status.Text = (string)proxy.Call("getStatus");
  }
  ...
}
```

이 예제에서 보듯이 ExternalInterfaceProxy 클래스의 `Call()` 메서드는 ActionScript의 `ExternalInterface.Call()`과 매우 비슷합니다. 첫 번째 매개 변수는 문자열로, 호출할 함수 이름입니다. 추가 매개 변수(여기에는 표시되지 않음)는 ActionScript 함수로 전달됩니다. ActionScript 함수가 값을 반환하면 앞의 예제와 같이 `Call()` 메서드에서 이 값을 반환합니다.

ExternalInterfaceProxy 클래스 내부

ActiveX 컨트롤 주위에 프록시 래퍼를 사용하는 것이 항상 유용하지는 않으며, 예를 들어, 다른 프로그래밍 언어로 또는 다른 플랫폼에 맞게 직접 proxy 클래스를 작성하려는 경우가 있을 수 있습니다. 프록시를 만드는 데 대한 모든 정보가 설명되어 있지는 않지만 이 예제에서는 proxy 클래스가 내부적으로 작동하는 방식을 이해할 수 있도록 설명합니다.

Shockwave Flash ActiveX 컨트롤의 `CallFunction()` 메서드를 사용하여 ActiveX 컨테이너에서 External API를 통해 ActionScript 함수를 호출합니다. ExternalInterfaceProxy 클래스의 `Call()` 메서드에서 추출한 다음 예제는 이 방법을 보여 줍니다.

```
// SWF Shockwave Flash ActiveX 컨트롤인 "_flashControl"에서
```

```
// ActionScript 함수를 호출합니다.
string response = _flashControl.CallFunction(request);
```

이 코드 인용문에서 `_flashControl`은 Shockwave Flash ActiveX 컨트롤입니다. `CallFunction()` 메서드를 사용하여 ActionScript 함수가 호출됩니다. 이 메서드는 하나의 매개 변수(이 예제에서는 `request`)를 사용합니다. 이 매개 변수는 호출할 ActionScript 함수 이름 및 모든 매개 변수 등 XML 형식의 명령을 포함하는 문자열입니다. ActionScript의 반환값은 XML 형식의 문자열로 인코딩된 다음 `CallFunction()` 호출의 반환값으로 다시 전송됩니다. 이 예제에서는 이 XML 문자열이 `response` 변수에 저장됩니다.

ActionScript에서 함수 호출을 수신하는 것은 여러 단계의 프로세스입니다. ActionScript에서 함수가 호출되면 Shockwave Flash ActiveX 컨트롤이 해당 FlashCall 이벤트를 전달하므로 SWF 파일에서 호출을 수신하려는 클래스(예: ExternalInterfaceProxy 클래스)는 해당 이벤트의 핸들러를 정의해야 합니다. ExternalInterfaceProxy 클래스에서 이벤트 핸들러 함수의 이름이 `_flashControl_FlashCall()`로 지정되고 다음과 같이 클래스 생성자에서 이벤트를 수신할 수 있도록 등록됩니다.

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
  _flashControl = flashControl;
  _flashControl.FlashCall += new
  _IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender,
  _IShockwaveFlashEvents_FlashCallEvent e)
{
  // 이벤트 객체의 request 속성("e.request")을 사용하여
  // 특정 액션을 실행합니다.
  ...
  // ActionScript에 값을 반환합니다.
  // 반환 값은 먼저 XML 형식의 문자열로 인코딩되어야 합니다.
  _flashControl.SetReturnValue(encodedResponse);
}
```

이벤트 객체(e)에는 함수 이름과 매개 변수 등 함수 호출에 대한 XML 형식의 정보를 포함하는 문자열인 `request` 속성(`e.request`)이 있습니다. 컨테이너에서는 이 정보를 사용하여 실행할 코드를 결정합니다. ExternalInterfaceProxy 클래스에서는 요청이 XML 형식에서 ExternalInterfaceCall 객체로 변환되어 같은 정보를 액세스하기 쉬운 형식으로 제공합니다. ActiveX 컨트롤의 `SetReturnValue()` 메서드는 ActionScript 호출자에게 함수 결과를 반환하는 데 사용됩니다. 마찬가지로 결과 매개 변수를 External API에서 사용하는 XML 형식으로 인코딩해야 합니다.

Shockwave Flash ActiveX 컨트롤을 호스팅하는 ActionScript와 응용 프로그램 간의 통신은 특정 XML 형식을 사용하여 함수 호출과 값을 인코딩합니다. Introvert IM C# 예제에서는 ExternalInterfaceProxy 클래스를 통해 응용 프로그램 양식의 코드가 ActionScript에 대해 송수신된 값을 직접 연산하고 Flash Player에서 사용되는 세부 XML 형식을 무시하도록 할 수 있습니다. 그렇게 하기 위해 ExternalInterfaceProxy 클래스는 ExternalInterfaceSerializer 클래스의 메서드를 사용하여 XML 메시지를 실제로 .NET 객체로 변환합니다. ExternalInterfaceSerializer 클래스에는 다음 네 가지 공용 메서드가 있습니다.

- `EncodeInvoke()`: 함수 이름 및 인수 C# ArrayList를 적절한 XML 형식으로 인코딩합니다.
- `EncodeResult()`: 결과 값을 적절한 XML 형식으로 인코딩합니다.
- `DecodeInvoke()`: ActionScript에서 함수 호출을 디코딩합니다. FlashCall 이벤트 객체의 `request` 속성은 `DecodeInvoke()` 메서드로 전달되고, 이는 ExternalInterfaceCall 객체에 대한 호출로 변환됩니다.
- `DecodeResult()`: ActionScript 함수를 호출한 결과 수신되는 XML을 디코딩합니다.

이러한 메서드는 C# 값을 External API의 XML 형식으로 인코딩하고 XML을 C# 객체로 디코딩합니다. Flash Player에서 사용하는 XML 형식에 대한 자세한 내용은 691페이지의 “External API의 XML 형식”을 참조하십시오.

제 26 장

Flash Player 보안

26

보안 문제는 Adobe, 사용자, 웹 사이트 소유자 및 내용 개발자의 주요 관심사입니다. 이에 따라 Adobe Flash Player 9에는 이러한 사용자, 웹 사이트 소유자 및 내용 개발자를 보호하기 위해 일련의 보안 규칙과 컨트롤이 포함되어 있습니다. 이 장에서는 Flash 응용 프로그램을 개발할 때 Flash Player 보안 모델을 사용하는 방법에 대해 설명합니다. 여기에 언급된 모든 SWF 파일은 따로 명시하지 않는 한 ActionScript 3.0을 사용하여 제작되었으며 Flash Player 9 이상에서 실행되는 것으로 가정합니다.

이 장은 보안에 대한 개괄적 설명을 포함하며 특정 API를 사용한 상세 구현 정보, 사용 시나리오 또는 기타 세부 사항에 대해 포괄적으로 설명하기 위한 것은 아닙니다. Flash Player 보안 개념에 대한 자세한 내용은 www.adobe.com/go/fp9_0_security에서 *Flash Player 9 보안* 백서를 참조하십시오.

목차

Flash Player 보안 개요

대부분의 Flash Player 보안은 로드된 SWF 파일, 미디어 및 기타 에셋의 원래 도메인을 기반으로 합니다. www.example.com과 같은 특정 인터넷 도메인을 기반으로 하는 SWF 파일은 언제든지 해당 도메인의 모든 데이터에 액세스할 수 있습니다. 이러한 에셋은 *보안 샌드박스*라는 동일한 보안 그룹에 포함됩니다. 자세한 내용은 719페이지의 "보안 샌드박스"를 참조하십시오.

예를 들어, SWF 파일은 SWF 파일, 비트맵, 오디오, 텍스트 파일 및 자체 도메인의 모든 에셋을 로드할 수 있습니다. 또한 동일한 도메인에서 ActionScript 3.0으로 작성된 두 SWF 파일 간의 크로스 스크립팅은 항상 허용됩니다. *크로스 스크립팅*은 ActionScript를 사용하여 하나의 SWF 파일에서 다른 SWF 파일의 속성, 메서드 및 객체에 액세스하는 기능입니다.
ActionScript 3.0을 사용하여 작성된 SWF 파일과 ActionScript 3.0 이전 버전을 사용하여 작성된 SWF 파일 간의 크로스 스크립팅은 지원되지 않습니다. 단, LocalConnection 클래스를 사용하여 이러한 파일 간의 통신은 가능합니다. 자세한 내용은 728페이지의 "크로스 스크립팅"을 참조하십시오.

기본적으로 다음과 같은 기본 보안 규칙이 적용됩니다.

- 동일한 보안 샌드박스의 리소스는 항상 서로 액세스할 수 있습니다.
- 원격 샌드박스의 SWF 파일에서는 로컬 파일과 데이터에 액세스할 수 없습니다.

Flash Player는 다음을 개별 도메인으로 간주하고 각각에 대해 개별 보안 샌드박스를 설정합니다.

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

http://example.com과 같은 이름을 가진 도메인이 http://192.0.34.166 등의 특정 IP 주소로 매핑되는 경우에도 각각에 대해 개별 보안 샌드박스가 설정됩니다.

SWF 파일에서 해당 샌드박스가 아닌 다른 샌드박스의 에셋에 액세스할 수 있도록 하기 위해 개발자는 다음과 같은 기본적인 두 가지 메서드를 사용할 수 있습니다.

- `Security.allowDomain()` 메서드(717페이지의 "제작자(개발자) 컨트롤" 참조)
- 크로스 도메인 정책 파일(714페이지의 "웹 사이트 컨트롤(크로스 도메인 정책 파일)" 참조)

다른 도메인에서 ActionScript 3.0 SWF 파일을 크로스 스크립팅하거나 다른 도메인의 데이터를 로드하는 SWF 파일 기능은 기본적으로 사용할 수 없습니다. 단, 로드된 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하면 이 기능을 사용할 수 있습니다. 자세한 내용은 728페이지의 "크로스 스크립팅"을 참조하십시오.

Flash Player 보안 모델에서는 *내용*을 로드하는 것과 *데이터*를 액세스하거나 로드하는 것을 구분합니다.

- 내용 로드 - *내용*이 시각적 미디어를 포함한 미디어로 정의된 경우 Flash Player는 오디오, 비디오 또는 표시된 미디어를 포함하는 SWF 파일을 표시합니다. *데이터*는 ActionScript 코드에만 액세스할 수 있는 항목으로 정의됩니다. Loader, Sound 및 NetStream과 같은 클래스를 사용하여 내용을 로드할 수 있습니다.
- 데이터 또는 데이터 로드로 내용 액세스 - 로드된 미디어 내용에서 데이터를 추출하거나 XML 파일 등의 외부 파일에서 데이터를 직접 로드하는 두 가지 방법을 사용하여 데이터에 액세스할 수 있습니다. 비트맵 객체, `BitmapData.draw()` 메서드, `Sound.id3` 속성 또는 `SoundMixer.computeSpectrum()` 메서드를 사용하여 로드된 미디어에서 데이터를 추출할 수 있습니다. URLStream, URLLoader, Socket 및 XMLSocket 등의 클래스를 사용하여 데이터를 로드할 수 있습니다.

Flash Player 보안 모델은 내용 로드와 데이터 액세스에 대해 서로 다른 규칙을 정의합니다. 일반적으로 데이터에 액세스하는 것보다 내용을 로드하는 데 적용되는 제한이 적습니다.

일반적으로, SWF 파일, 비트맵, mp3 파일, 비디오 등의 내용은 모든 위치에서 로드될 수 있지만 로드하는 SWF 파일의 도메인이 아닌 다른 도메인의 내용은 별도의 보안 샌드박스로 구분됩니다.

내용을 로드하는 데는 다음과 같은 몇 가지 제한이 있습니다.

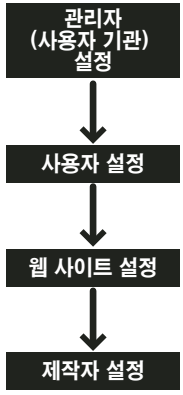
- 기본적으로 사용자의 하드 드라이브와 같이 네트워크 주소가 아닌 위치에서 로드된 로컬 SWF 파일은 local-with-filesystem 샌드박스로 분류됩니다. 이러한 파일은 네트워크에서 내용을 로드할 수 없습니다. 자세한 내용은 719페이지의 “로컬 샌드박스”를 참조하십시오.
- RTMP(Real-Time Messaging Protocol) 서버로 내용에 대한 액세스를 제한할 수 있습니다. 자세한 내용은 728페이지의 “RTMP 서버를 사용하여 제공된 내용”을 참조하십시오.

로드된 미디어가 이미지, 오디오, 비디오 또는 해당 데이터(픽셀 데이터 및 사운드 데이터 등)인 경우, SWF 파일의 도메인이 미디어의 원래 도메인에 있는 크로스 도메인 정책 파일에 포함된 경우가 아닌 한, 해당 보안 샌드박스 외부에 있는 SWF 파일에서 로드된 미디어에 액세스할 수 없습니다. 자세한 내용은 732페이지의 “데이터로 로드된 미디어 액세스”를 참조하십시오.

다른 형식의 로드된 데이터에는 URLLoader 객체를 통해 로드되는 텍스트나 XML 파일이 있습니다. 이 경우에도 다른 보안 샌드박스에서 모든 데이터에 액세스하려면 원래 도메인에 있는 크로스 도메인 정책 파일을 통해 액세스 권한이 부여되어야 합니다. 자세한 내용은 735페이지의 “URLLoader 및 URLStream 사용”을 참조하십시오.

권한 컨트롤 개요

Flash Player 클라이언트 런타임 보안 모델은 리소스를 중심으로 설계되어 있으며, 이러한 리소스는 SWF 파일, 로컬 데이터 및 인터넷 URL 등의 객체입니다. 이러한 리소스를 소유하거나 사용하는 *관계자*는 자신이 소유한 리소스에 대해 컨트롤(보안 설정)을 실행할 수 있습니다. 각 리소스에는 네 명의 관계자가 있습니다. Flash Player에서는 다음 그림과 같이 컨트롤에 대해 권한 계층 구조를 엄격하게 적용합니다.



보안 컨트롤 계층 구조

예를 들어, 관리자가 리소스에 대한 액세스를 제한하면 다른 관계자가 해당 제한을 무시할 수 없습니다.

관리자, 사용자 및 웹 사이트 컨트롤은 다음 단원에서 자세히 설명합니다. 제작자(개발자) 설정은 이 장의 뒷부분에 설명되어 있습니다.

관리자 컨트롤

컴퓨터에 관리자 권한으로 로그인한 관리자의 경우 해당 컴퓨터의 다른 모든 사용자에게 영향을 주는 Flash Player 보안 설정을 적용할 수 있습니다. 가정용 컴퓨터와 같이 엔터프라이즈 환경이 아닌 컴퓨터에는 보통 한 명의 사용자가 관리자 권한도 가지고 있습니다. 엔터프라이즈 환경에서도 개별 사용자가 컴퓨터에 대해 관리자 권한을 가질 수 있습니다.

관리자 컨트롤에는 다음 두 가지 유형이 있습니다.

- mms.cfg 파일
- Global Flash Player Trust 디렉토리

mms.cfg 파일

mms.cfg 파일은 Mac OS X 시스템인 경우 /Library/Application Support/Macromedia/mms.cfg에 있고, Microsoft Windows 시스템인 경우 시스템 디렉토리의 Macromedia Flash Player 폴더에 있습니다(예: 기본 Windows XP 설치인 경우 C:\windows\system32\macromed\flash\mms.cfg).

Flash Player를 시작하면 이 파일의 보안 설정을 읽어 기능을 제한하는 데 사용합니다.

mms.cfg 파일에는 다음과 같은 작업을 수행하기 위해 관리자가 사용하는 설정이 포함되어 있습니다.

- **데이터 로드** - 로컬 SWF 파일의 읽기를 제한하고 파일 다운로드 및 업로드를 비활성화하며 영구 공유 객체에 대한 저장 제한을 설정합니다.
- **개인 정보 제어** - 마이크 및 카메라 액세스를 비활성화하고, SWF 파일에서 윈도우 없는 내용을 실행하지 않도록 하며, 브라우저 윈도우에 표시된 URL과 일치하지 않는 도메인의 SWF 파일에서 영구 공유 객체에 액세스하지 못하도록 합니다.
- **Flash Player 업데이트** - Flash Player의 업데이트 버전을 확인하는 간격을 설정하고, Flash Player 업데이트 정보를 확인하고 해당 업데이트 버전을 다운로드할 URL을 지정하며, Flash Player 전체에 대한 자동 업데이트를 비활성화합니다.
- **이전 파일 지원** - local-trusted 샌드박스에 이전 버전의 SWF 파일을 배치할지 여부를 지정합니다.
- **로컬 파일 보안** - 로컬 파일을 local-trusted 샌드박스에 배치할 수 있는지 여부를 지정합니다.
- **전체 화면 모드** - 전체 화면 모드를 비활성화합니다.

SWF 파일은 `Capabilities.avHardwareDisable` 및 `Capabilities.localFileReadDisable` 속성을 호출하여 비활성화된 기능에 대한 일부 정보를 액세스할 수 있습니다. 그러나 mms.cfg 파일의 설정 대부분은 ActionScript에서 쿼리할 수 없습니다.

컴퓨터에 응용 프로그램과 무관한 보안 및 개인 정보 보호 정책을 적용하려면 시스템 관리자가 mms.cfg 파일을 수정해야 합니다. 응용 프로그램의 설치 관리자에서는 mms.cfg 파일을 사용할 수 없습니다. 관리자 권한으로 실행되는 설치 관리자는 mms.cfg 파일의 내용을 수정할 수 있지만, Adobe는 이러한 사용을 사용자의 신뢰 위반으로 간주하고 설치 프로그램 작성자에게 mms.cfg 파일을 수정하지 않도록 권고합니다.

Global Flash Player Trust 디렉토리

관리자 및 설치 프로그램은 지정된 로컬 SWF 파일을 신뢰할 수 있는 파일로 등록할 수 있습니다. 이러한 SWF 파일은 local-trusted 샌드박스에 할당되며, 다른 SWF 파일과 상호 작용하고 원격이나 로컬의 모든 위치에서 데이터를 로드할 수 있습니다. 파일은 다음 위치에서 mms.cfg 파일과 동일한 디렉토리에 있는 Global Flash Player Trust 디렉토리에 신뢰할 수 있는 파일로 지정됩니다(위치는 현재 사용자에 따라 다름).

- Windows: system\Macromed\Flash\FlashPlayerTrust

  (예: C:\windows\system32\Macromed\Flash\FlashPlayerTrust)
- Mac: app support/Macromedia/FlashPlayerTrust

  (예: /Library/Application Support/Macromedia/FlashPlayerTrust)

Flash Player Trust 디렉토리에는 텍스트 파일을 무제한으로 포함할 수 있으며 각 텍스트 파일에는 신뢰할 수 있는 경로가 한 줄에 하나씩 나열되어 있습니다. 각 경로는 개별 SWF 파일, HTML 파일 또는 디렉토리일 수 있습니다. 주석 행은 # 심볼로 시작됩니다. 예를 들어, 다음과 같은 텍스트가 포함된 Flash Player 신뢰 구성 파일은 지정된 디렉토리 및 모든 하위 디렉토리의 모든 파일에 대해 신뢰 상태를 부여합니다.

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

신뢰 구성 파일에 나열된 경로는 항상 로컬 경로나 SMB 네트워크 경로여야 합니다. 신뢰 구성 파일의 모든 HTTP 경로는 무시되며, 로컬 파일만 신뢰할 수 있습니다.

충돌을 피하려면 각 신뢰 구성 파일에 설치 응용 프로그램에 해당하는 파일 이름을 부여하고 .cfg 파일 확장명을 사용합니다.

설치 프로그램을 통해 로컬로 실행되는 SWF 파일을 배포하려는 개발자의 경우, 설치 프로그램에서 Global Flash Player Trust 디렉토리에 구성 파일을 추가하도록 함으로써, 배포하는 파일에 전체 권한을 부여할 수 있습니다. 이때, 설치 프로그램은 관리자 권한이 있는 사용자가 실행해야 합니다. mms.cfg 파일과 달리 Global Flash Player Trust 디렉토리는 신뢰 권한을 부여하기 위한 목적으로 설치 프로그램에 포함될 수 있습니다. 관리자와 설치 프로그램 모두 Global Flash Player Trust 디렉토리를 사용하여 신뢰할 수 있는 로컬 응용 프로그램을 지정할 수 있습니다.

또한 개별 사용자를 위한 Flash Player Trust 디렉토리도 있습니다(다음 단원, "사용자 컨트롤" 참조).

사용자 컨트롤

Flash Player에서는 권한 설정에 대해 설정 UI, 설정 관리자 및 User Flash Player Trust 디렉토리의 서로 다른 세 가지의 사용자 수준 메커니즘을 제공합니다.

설정 UI 및 설정 관리자

설정 UI는 특정 도메인의 설정을 구성하는 빠른 대화형 메커니즘입니다. 설정 관리자는 보다 자세한 인터페이스와 많은 도메인이나 전체 도메인의 권한에 영향을 주는 글로벌 변경 기능을 제공합니다. 또한 SWF 파일에서 새로운 권한을 요청할 때 보안 또는 개인 정보에 대해 런타임 의사 결정이 필요한 경우, 사용자가 일부 Flash Player 설정을 조정할 수 있는 대화 상자가 표시됩니다.

설정 관리자 및 설정 UI에서는 다음과 같은 보안 관련 옵션을 제공합니다.

- 카메라 및 마이크 설정 - 사용자가 컴퓨터에서 카메라 및 마이크에 대한 Flash Player 액세스를 제어할 수 있습니다. 사용자가 전체 사이트 또는 특정 사이트에 대한 액세스를 허용하거나 거부할 수 있습니다. 사용자가 전체 사이트나 특정 사이트에 대한 설정을 지정하지 않은 경우, SWF 파일에서 카메라나 마이크에 액세스하려고 하면, SWF 파일에서 해당 장치에 액세스하도록 허용할지 여부를 선택할 수 있는 대화 상자가 표시됩니다. 이때, 사용자는 사용할 카메라나 마이크를 지정하고 마이크의 민감도를 설정할 수 있습니다.
- 공유 객체 저장 설정 - 도메인에서 영구 공유 객체 저장에 사용할 디스크 공간을 선택할 수 있습니다. 사용자는 특정 도메인에 대해 임의의 값을 설정하고 새로운 도메인에 대해 기본 설정을 지정할 수 있습니다. 기본 디스크 공간 제한 값은 100KB입니다. 영구 공유 객체에 대한 자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 SharedObject 클래스를 참조하십시오.

참고 **mms.cfg 파일에 지정된 모든 설정(710페이지의 "관리자 컨트롤" 참조)은 설정 관리자에 반영되지 않습니다.**

설정 관리자에 대한 자세한 내용은 www.adobe.com/go/settingsmanager_kr을 참조하십시오.

User Flash Player Trust 디렉토리

사용자 및 설치 프로그램은 지정된 로컬 SWF 파일을 신뢰할 수 있는 파일로 등록할 수 있습니다. 이러한 SWF 파일은 local-trusted 샌드박스에 할당되며, 다른 SWF 파일과 상호 작용하고 원격이나 로컬의 모든 위치에서 데이터를 로드할 수 있습니다. 사용자는 다음 위치에서 Flash 공유 객체 저장 영역과 동일한 디렉토리에 있는 User Flash Player Trust 디렉토리에 신뢰할 수 있는 파일로 지정합니다(위치는 현재 사용자에 따라 다름).

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (예: C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust)

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (예: /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

이러한 설정은 현재 사용자에게만 영향을 주며 컴퓨터에 로그인한 다른 사용자에게는 적용되지 않습니다. 관리자 권한이 없는 사용자가 시스템의 자체 소유 부분에 응용 프로그램을 설치한 경우, 설치 관리자에서 User Flash Player Trust 디렉토리에 이 응용 프로그램을 해당 사용자에 대해 신뢰할 수 있는 것으로 등록할 수 있습니다.

설치 프로그램을 통해 로컬로 실행되는 SWF 파일을 배포하는 개발자의 경우, 설치 프로그램에서 User Flash Player Trust 디렉토리에 구성 파일을 추가하도록 함으로써 배포하는 파일에 전체 권한을 부여할 수 있습니다. 이런 경우에도 User Flash Player Trust 디렉토리 파일이 사용자 컨트롤로 간주되며, 이는 사용자 액션(설치)으로 시작되기 때문입니다.

또한 관리자나 설치 프로그램에서 Global Flash Player Trust 디렉토리를 사용하여 컴퓨터의 모든 사용자에 대해 응용 프로그램을 등록할 수 있습니다(710페이지의 "관리자 컨트롤" 참조).

웹 사이트 컨트롤(크로스 도메인 정책 파일)

웹 서버의 데이터를 다른 도메인의 SWF 파일에서 사용할 수 있도록 하기 위해 사용자 서버에 크로스 도메인 정책 파일을 작성할 수 있습니다. *크로스 도메인 정책 파일*은 특정 또는 모든 도메인에서 제공되는 SWF 파일에서 해당 서버의 데이터나 문서를 사용할 수 있음을 나타내기 위해 사용되는 XML 파일입니다. 서버의 정책 파일에 지정된 도메인에서 제공되는 SWF 파일은 해당 서버의 데이터나 에셋에 대한 액세스가 허용됩니다.

크로스 도메인 정책 파일은 다음을 포함하여 많은 에셋에 대한 액세스에 영향을 줍니다.

- 비트맵, 사운드 및 비디오 데이터
- XML 및 텍스트 파일 로드
- 소켓 및 XML 소켓 연결에 대한 액세스
- 다른 보안 도메인의 SWF 파일을 로드하는 SWF 파일의 보안 도메인으로 가져오기

자세한 사항은 이 장의 뒷부분에 설명되어 있습니다.

정책 파일 구문

다음 예제는 `*.example.com`, `www.friendOfExample.com` 및 `192.0.34.166`에서 시작되는 SWF 파일에 대한 액세스를 허용하는 정책 파일을 보여 줍니다.

```
<?xml version="1.0"?>
<cross-domain-policy>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="www.friendOfExample.com" />
  <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

SWF 파일에서 다른 도메인의 데이터에 액세스하려고 하면 Flash Player는 자동으로 해당 도메인에서 정책 파일을 로드하려고 시도합니다. 액세스하려는 SWF 파일의 도메인이 정책 파일에 포함되어 있으면 자동적으로 데이터에 액세스할 수 있게 됩니다.

기본적으로 `crossdomain.xml`이라는 이름의 정책 파일이 해당 서버의 루트 디렉토리에 상주해야 합니다. 하지만 SWF 파일에서 `Security.loadPolicyFile()` 메서드를 호출하여 다른 이름 또는 다른 디렉토리 위치를 확인할 수 있습니다. 크로스 도메인 정책 파일은 정책 파일이 로드되어 있는 디렉토리와 해당 하위 디렉토리에만 적용됩니다. 따라서 루트 디렉토리에 있는 정책 파일은 전체 서버에 적용되지만 임의의 하위 디렉토리에서 로드된 정책 파일은 해당 디렉토리와 하위 디렉토리에만 적용됩니다.

정책 파일은 해당 파일이 상주하는 특정 서버에 대한 액세스에만 영향을 줍니다. 예를 들어, https://www.adobe.com:8080/crossdomain.xml에 있는 정책 파일은 8080 포트에서 HTTPS를 통해 www.macromedia.com에 수행된 데이터 로드 호출에만 적용됩니다.

크로스 도메인 정책 파일에는 0개 이상의 `<allow-access-from>` 태그가 포함된 단일 `<cross-domain-policy>` 태그가 들어 있습니다. 각 `<allow-access-from>` 태그에는 `domain`이라는 속성이 있습니다. 이 속성은 정확한 IP 주소, 정확한 도메인 또는 와일드카드 도메인(임의의 도메인)을 지정합니다. 와일드카드 도메인은 모든 도메인과 IP 주소를 나타내는 경우 단일 별표(*) 또는 특정 접미어로 끝나는 도메인을 나타내는 경우 접미어가 뒤에 붙은 별표로 표현됩니다. 접미어는 점으로 시작해야 합니다. 그러나, 접미어가 붙은 와일드카드 도메인은 앞에 오는 점을 제외한 접미어만으로 구성되는 도메인을 나타낼 수 있습니다. 예를 들어, foo.com은 *.foo.com에 속한 것으로 생각할 수 있습니다. 와일드카드는 IP 도메인 형식에 사용할 수 없습니다.

IP 주소를 지정하면 IP 구문(예: http://65.57.83.12/flashmovie.swf)을 사용하여 해당 IP 주소에서 로드된 SWF에만 액세스 권한이 부여됩니다. 이 경우 도메인 이름 구문을 사용하여 로드된 SWF에는 액세스 권한이 부여되지 않습니다. Flash Player는 DSN 이름 확인을 수행하지 않습니다.

다음 예제와 같이 모든 도메인의 문서에 액세스할 수 있도록 허용할 수 있습니다.

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

각 `<allow-access-from>` 태그에는 선택 사항인 `secure` 속성이 포함될 수 있으며 기본값은 `true`입니다. 정책 파일이 HTTPS 서버에 있을 경우 HTTPS가 아닌 서버의 SWF 파일이 HTTPS 서버에서 데이터를 로드할 수 있도록 허용하려면 이 속성을 `false`로 설정합니다.

`secure` 속성을 `false`로 설정하면 HTTPS에서 제공하는 보안 기능이 약화될 수 있습니다. 특히 이 속성을 `false`로 설정하면 보안 내용이 열려서 스누핑(snooping) 및 스푸핑(spoofing) 공격에 취약해집니다. 따라서 `secure` 속성을 `false`로 설정하지 않아야 합니다.

로드할 데이터가 HTTPS 서버에 있고, 해당 데이터를 로드하는 SWF 파일이 HTTP 서버에 있는 경우에는 로드하는 SWF 파일을 HTTPS 서버로 이동하여 HTTPS의 보호 하에 보안 데이터의 모든 복사본을 보관하는 것이 좋습니다. 그러나 로드하는 SWF 파일을 HTTP 서버에 두기로 결정한 경우에는 다음 코드에 표시된 대로 `secure="false"` 속성을 `<allow-access-from>` 태그에 추가합니다.

```
<allow-access-from domain="www.example.com" secure="false" />
```

정책 파일에 `<allow-access-from>` 태그가 없으면 서버에 정책이 없는 것과 동일한 결과가 발생합니다.

소켓 정책 파일

ActionScript 객체는 문서 기반 서버 연결 및 소켓 연결의 서로 다른 두 종류의 서버 연결을 인스턴스화합니다. Loader, Sound, URLLoader 및 URLStream과 같은 ActionScript 객체는 문서 기반 연결을 인스턴스화하며 각각은 URL에서 파일을 로드합니다. ActionScript Socket 및 XMLSocket 객체는 소켓 연결을 만들어 로드된 문서가 아니라 스트리밍 데이터로 작업합니다. Flash Player에서는 문서 기반 정책 파일과 소켓 정책 파일의 두 종류의 정책 파일을 지원합니다. 문서 기반 연결에는 문서 기반 정책 파일이 필요하고 소켓 연결에는 소켓 정책 파일이 필요합니다.

Flash Player에서는 정책 파일을 전송할 때 연결 시도에서 사용하려는 프로토콜과 동일한 프로토콜을 사용해야 합니다. 예를 들어, 정책 파일이 HTTP 서버에 있는 경우 다른 도메인의 SWF 파일은 HTTP 서버로 데이터를 로드할 수 있습니다. 하지만 동일한 서버에 소켓 파일을 제공하지 않으면 다른 도메인의 SWF 파일에서 소켓 수준의 서버에 연결할 수 없게 됩니다. 소켓 정책 파일을 가져올 때는 연결할 때와 같은 방법을 사용해야 합니다.

소켓 서버에서 제공하는 정책 파일은 액세스가 허용되는 포트를 지정해야 한다는 점을 제외하고 다른 정책 파일과 동일한 구문을 사용합니다. 1024보다 낮은 포트에서 제공되는 정책 파일에서는 모든 포트에 액세스를 허용할 수 있고, 1024 이상의 포트에서 제공되는 정책 파일에서는 1024 이상의 포트에만 액세스를 허용할 수 있습니다. 허용되는 포트는 `<allow-access-from>` 태그의 `to-ports` 속성에 지정됩니다. 단일 포트 번호, 포트 범위 및 와일드카드가 모두 허용되는 값입니다.

XMLSocket 정책 파일의 예는 다음과 같습니다.

```
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example2.com" to-ports="516-523" />
   <allow-access-from domain="www.example2.com" to-ports="507,516-523" />
   <allow-access-from domain="www.example3.com" to-ports="*" />
</cross-domain-policy>
```

정책 파일이 처음으로 Flash Player 6에 도입되었을 때는 소켓 정책 파일이 지원되지 않았습니다. 따라서 소켓 서버에 대한 연결은 소켓 서버와 동일한 호스트의 포트 80에 있는 HTTP 서버의 크로스 도메인 정책 파일의 기본 위치에 있는 정책 파일에 의해 허가되었습니다. Flash Player 9에서는 기존의 서버 연결 방식을 보존하기 위해 이 기능을 지원합니다. 그러나 이제 Flash Player에서는 기본적으로 소켓 연결과 동일한 포트에 있는 소켓 정책 파일을 가져옵니다. HTTP 기반 정책 파일을 사용하여 소켓 연결을 허가하려면 다음과 같은 코드를 사용하여 명시적으로 HTTP 정책 파일을 요청해야 합니다.

```
Security.loadPolicyFile("http://socketServerHost.com/crossdomain.xml")
```

또한 소켓 연결을 허가하려면 HTTP 정책 파일을 다른 HTTP 위치가 아닌 크로스 도메인 정책 파일의 기본 위치에서 가져와야 합니다. HTTP 서버에서 가져온 정책 파일은 암시적으로 1024 이상의 모든 포트에 대해 소켓 액세스를 허가하며 HTTP 정책 파일의 `to-ports` 속성은 무시됩니다.

소켓 정책 파일에 대한 자세한 내용은 735페이지의 "소켓 연결"을 참조하십시오.

정책 파일 미리 로드

서버에서 데이터를 로드하고 소켓에 연결하는 것은 비동기식 작업으로, Flash Player에서는 단순히 주요 작업을 시작하기 전에 크로스 도메인 정책 파일이 다운로드되기를 기다립니다. 그러나 이미지에서 픽셀 데이터를 추출하거나 사운드에서 샘플 데이터를 추출하는 것은 동기식 작업으로, 데이터를 추출하기 전에 크로스 도메인 파일을 먼저 로드해야 합니다. 미디어를 로드하는 경우에는 다음과 같이 미디어에서 크로스 도메인 정책 파일을 확인하도록 지정해야 합니다.

- `Loader.load()` 메서드를 사용하는 경우에는 LoaderContext 객체인 `context` 매개 변수의 `checkPolicyFile` 속성을 설정합니다.
- `<img>` 태그를 사용하여 텍스트 필드에 이미지를 포함하는 경우, `<img checkPolicyFile = "true" src = "example.jpg">`와 같이 `<img>` 태그의 `checkPolicyFile` 속성을 `"true"`로 설정합니다.
- `Sound.load()` 메서드를 사용하는 경우에는 SoundLoaderContext 객체인 `context` 매개 변수의 `checkPolicyFile` 속성을 설정합니다.
- NetStream 클래스를 사용하는 경우, NetStream 객체의 `checkPolicyFile` 속성을 설정합니다.

이 매개 변수 중 하나를 설정하는 경우에는 Flash Player에서 먼저 해당 도메인에서 이미 다운로드한 모든 정책 파일을 확인합니다. 그리고 나서 `Security.loadPolicyFile()` 메서드에 대한 모든 대기 중인 호출을 검토하여 해당 범위 내에 있는지 확인하고 해당하는 경우 기다립니다. 그런 다음 서버의 기본 위치에서 크로스 도메인 정책 파일을 찾습니다.

제작자(개발자) 컨트롤

보안 권한을 부여하는 데 사용되는 기본 ActionScript API는 `Security.allowDomain()` 메서드이며, 이 메서드를 사용하여 지정한 도메인의 SWF 파일에 권한을 부여합니다. 다음 예제에서는 SWF 파일이 www.example.com 도메인에서 제공되는 SWF 파일에 대한 액세스 권한을 부여합니다.

```
Security.allowDomain("www.example.com")
```

이 메서드는 다음과 같은 권한을 부여합니다.

- SWF 파일 간 크로스 스크립팅(728페이지의 “크로스 스크립팅” 참조)
- 표시 목록 액세스(731페이지의 “표시 목록 탐색” 참조)
- 이벤트 감지(731페이지의 “이벤트 보안” 참조)
- Stage 객체의 속성 및 메서드에 대한 전체 액세스(730페이지의 “스테이지 보안” 참조)

`Security.allowDomain()` 메서드를 호출하는 주요 목적은 도메인 외부에 있는 SWF 파일에 `Security.allowDomain()` 메서드를 호출하는 SWF 파일을 스크립팅할 수 있는 권한을 부여하는 것입니다. 자세한 내용은 728페이지의 “크로스 스크립팅”을 참조하십시오.

`Security.allowDomain()` 메서드에 매개 변수로 IP 주소를 지정하더라도 지정된 IP 주소에 있는 모든 항목에 액세스가 허용되는 것은 아닙니다. 이때 해당 IP 주소에 매핑되는 도메인 이름이 아닌 URL에 지정한 IP 주소가 들어 있는 항목에만 액세스가 허용됩니다. 예를 들어, 도메인 이름 www.example.com이 IP 주소 192.0.34.166으로 매핑되는 경우 `Security.allowDomain("192.0.34.166")`을 호출해도 www.example.com에 액세스가 허용되지 않습니다.

`"*"` 와일드카드를 `Security.allowDomain()` 메서드로 전달하여 모든 도메인에서의 액세스를 허용할 수 있습니다. 이렇게 하면 *모든* 도메인의 SWF 파일에 권한이 부여되어 호출하는 SWF 파일을 스크립팅하므로 `"*"` 와일드카드는 주의해서 사용해야 합니다.

ActionScript에는 `Security.allowInsecureDomain()`이라는 두 번째 권한 부여 API가 포함되어 있습니다. 이 메서드는 `Security.allowDomain()` 메서드와 동일한 작용을 하지만, 보안 HTTPS 연결에 의해 제공되는 SWF 파일에서 이 메서드를 호출하는 경우에 HTTP와 같이 비보안 프로토콜에서 제공되는 다른 SWF 파일에 대해서도 호출하는 SWF 파일에 대한 액세스를 추가로 허용한다는 것이 다릅니다. 그러나 보안 프로토콜(HTTPS)의 파일과 비보안 프로토콜(HTTP 등)의 파일 간에 스크립팅을 허용하는 것은 보안상 안전하지 않습니다. 이렇게 하면 보안 내용이 열려 스누핑 및 스푸핑 공격에 취약해집니다. 이러한 공격의 작동 방식은 다음과 같습니다. `Security.allowInsecureDomain()` 메서드는 HTTP 연결을 통해 제공되는 SWF 파일에 대해 보안 HTTPS 데이터에 대한 액세스를 허용하므로 공격자가 HTTP 서버와 사용자 간에 끼어 들어 HTTP SWF 파일을 자신의 파일로 교체함으로써 HTTPS 데이터에 액세스할 수 있게 됩니다.

또 다른 중요한 보안 관련 메서드에는 `Security.loadPolicyFile()` 메서드가 있으며, 이를 통해 Flash Player는 비표준 위치에서 크로스 도메인 정책 파일을 확인합니다. 자세한 내용은 714페이지의 “웹 사이트 컨트롤(크로스 도메인 정책 파일)”을 참조하십시오.

보안 샌드박스

클라이언트 컴퓨터는 외부 웹 사이트나 로컬 파일 시스템 등의 여러 소스에서 개별 SWF 파일을 얻습니다. Flash Player는 SWF 파일과 공유 객체, 비트맵, 사운드, 비디오 및 데이터 파일 등의 기타 리소스를, 이들이 Flash Player로 로드될 때의 원래 위치를 기준으로 하여 개별적으로 보안 샌드박스에 할당합니다. 다음 단원에서는 지정된 샌드박스에서 SWF 파일이 액세스할 수 있는 항목을 제어하는 Flash Player의 규칙에 대해 설명합니다.

보안 샌드박스에 대한 자세한 내용은 *Flash Player 9 보안* 백서를 참조하십시오.

원격 샌드박스

Flash Player는 인터넷의 SWF 파일을 포함하여 에셋을 해당 웹 사이트의 원래 도메인에 해당하는 별도의 샌드박스에 분류합니다. 기본적으로 이러한 파일은 해당 서버의 모든 리소스에 대한 액세스가 허용됩니다. 원격 SWF 파일은 크로스 도메인 정책 파일 및 `Security.allowDomain()` 메서드와 같은 명시적인 웹 사이트 및 제작자 권한을 사용하여 다른 도메인의 데이터에 추가로 액세스할 수 있습니다. 자세한 내용은 714페이지의 “웹 사이트 컨트롤(크로스 도메인 정책 파일)” 및 717페이지의 “제작자(개발자) 컨트롤”을 참조하십시오.

원격 SWF 파일에서는 로컬 파일이나 리소스를 로드할 수 없습니다.

자세한 내용은 *Flash Player 9 보안* 백서를 참조하십시오.

로컬 샌드박스

*로컬 파일*은 `file:` 프로토콜이나 UNC(Universal Naming Convention) 경로를 사용하여 참조되는 모든 파일을 의미합니다. 로컬 SWF 파일은 다음 세 로컬 샌드박스 중 하나에 배치됩니다.

- local-with-filesystem 샌드박스 - 보안을 위해 Flash Player는 기본적으로 모든 로컬 SWF 파일 및 에셋을 local-with-file-system 샌드박스에 배치합니다. 이 샌드박스에서 SWF 파일은 URLLoader 클래스 등을 사용하여 로컬 파일을 읽을 수 있지만, 네트워크와는 어떤 방식으로도 통신할 수 없습니다. 따라서 사용자는 로컬 데이터가 네트워크로 누출되거나 다른 방식으로 부적절하게 공유되지 않는다는 확신을 가질 수 있습니다.

- local-with-networking 샌드박스 - SWF 파일을 컴파일할 때, 로컬 파일로 실행되지만 네트워크 액세스가 가능하도록 지정할 수 있습니다(720페이지의 "로컬 SWF 파일의 샌드박스 유형 설정" 참조). 이러한 파일은 local-with-networking 샌드박스에 배치됩니다. local-with-networking 샌드박스에 할당된 SWF 파일에서는 해당 로컬 파일에 액세스할 수 없습니다. 대신, SWF 파일은 네트워크의 데이터에 액세스할 수 있습니다. 하지만 크로스 도메인 정책 파일이나 `Security.allowDomain()` 메서드에 대한 호출을 통해 권한이 부여되지 않는 한 local-with-networking SWF 파일은 여전히 네트워크의 데이터를 읽을 수 없습니다. 이러한 권한을 부여하려면 크로스 도메인 정책 파일에서 `<allow-access-from domain="*"/>` 또는 `Security.allowDomain("*")`을 사용하여 *모든* 도메인에 권한을 부여해야 합니다. 자세한 내용은 714페이지의 "웹 사이트 컨트롤(크로스 도메인 정책 파일)" 및 717페이지의 "제작자(개발자) 컨트롤"을 참조하십시오.
- local-trusted 샌드박스 - 사용자나 설치 프로그램에 의해 신뢰할 수 있는 파일로 등록된 로컬 SWF 파일은 local-trusted 샌드박스에 배치됩니다. 시스템 관리자와 사용자는 보안 고려 사항에 따라 local-trusted 샌드박스에 또는 해당 샌드박스로부터 로컬 SWF 파일을 재할당하거나 이동할 수도 있습니다(710페이지의 "관리자 컨트롤" 및 712페이지의 "사용자 컨트롤" 참조). local-trusted 샌드박스에 할당된 SWF 파일은 다른 SWF 파일과 상호 작용하고 원격이나 로컬의 모든 위치에서 데이터를 로드할 수 있습니다.

local-with-networking과 local-with-filesystem 샌드박스 간의 통신은 물론 local-with-filesystem과 원격 샌드박스 간의 통신이 엄격하게 금지됩니다. Flash 응용 프로그램이나 사용자 또는 관리자는 이러한 통신을 허용하는 권한을 부여할 수 없습니다.

어느 방향으로든 로컬 HTML 파일과 로컬 SWF 파일 간에 스크립팅하려면(예: ExternalInterface 클래스 사용) 관련 HTML 파일과 로컬 SWF 파일이 모두 local-trusted 샌드박스에 있어야 합니다. 이것은 브라우저의 로컬 보안 모델이 Flash Player 로컬 보안 모델과 다르기 때문입니다.

local-with-networking 샌드박스의 SWF 파일은 local-with-filesystem 샌드박스의 SWF 파일을 로드할 수 없습니다. local-with-filesystem 샌드박스의 SWF 파일은 local-with-networking 샌드박스의 SWF 파일을 로드할 수 없습니다.

로컬 SWF 파일의 샌드박스 유형 설정

Adobe Flash CS3 Professional 제작 도구에서 문서의 제작 설정을 설정함으로써 local-with-filesystem 샌드박스 또는 local-with-networking 샌드박스에 대해 SWF 파일을 구성할 수 있습니다. 자세한 내용은 *Flash 사용 설명서*에서 "Flash SWF 파일 포맷에 대한 제작 옵션 설정"을 참조하십시오.

컴퓨터의 최종 사용자나 관리자는 로컬 SWF 파일을 신뢰할 수 있는 파일로 지정하여 해당 파일에서 로컬과 네트워크의 모든 도메인에 있는 데이터를 로드하도록 할 수 있습니다. 이것은 Global Flash Player Trust 및 User Flash Player Trust 디렉토리에 지정됩니다. 자세한 내용은 710페이지의 "관리자 컨트롤" 및 712페이지의 "사용자 컨트롤"을 참조하십시오.

로컬 샌드박스에 대한 자세한 내용은 719페이지의 "로컬 샌드박스"를 참조하십시오.

Security.sandboxType 속성

SWF 파일의 제작자는 읽기 전용의 정적 `Security.sandboxType` 속성을 사용하여 SWF 파일에 할당된 샌드박스의 유형을 확인할 수 있습니다. Security 클래스에는 다음과 같이 `Security.sandboxType` 속성의 가능한 값을 나타내는 상수가 포함되어 있습니다.

- `Security.REMOTE` - 인터넷 URL에서 가져온 SWF 파일이며 도메인 기반 샌드박스 규칙에 따라 작동합니다.
- `Security.LOCAL_WITH_FILE` - 로컬 SWF 파일이지만 사용자가 신뢰하지 않았고 네트워킹이 지정되지 않았습니다. 이 SWF 파일은 로컬 데이터 소스에서 읽을 수 있지만 인터넷과 통신할 수 없습니다.
- `Security.LOCAL_WITH_NETWORK` - 로컬 SWF 파일이고 사용자가 신뢰하지 않았지만 네트워킹이 지정되었습니다. 이 SWF 파일은 인터넷과 통신할 수 있지만 로컬 데이터 소스에서 읽을 수는 없습니다.
- `Security.LOCAL_TRUSTED` - 로컬 SWF 파일이고 설정 관리자나 Flash Player 신뢰 구성 파일을 사용하여 사용자가 신뢰한 파일입니다. 이 SWF 파일은 로컬 데이터 소스에서 읽을 수 있고 인터넷과 통신할 수 있습니다.

네트워킹 API 제한

SWF 내용을 포함하는 HTML 페이지의 `<object>` 및 `<embed>` 태그에서 `allowNetworking` 매개 변수를 설정하여 네트워크 기능에 대한 SWF 파일의 액세스를 제어할 수 있습니다.

`allowNetworking`의 가능한 값은 다음과 같습니다.

- `"all"`(기본값) - SWF에서 모든 네트워크 API가 허용됩니다.
- `"internal"` - SWF 파일에서 이 단원의 뒷부분에 나열되어 있는 브라우저 내비게이션이나 브라우저 상호 작용 API를 호출할 수 없지만 다른 네트워킹 API를 호출할 수 있습니다.
- `"none"` - SWF 파일에서 이 단원의 뒷부분에 나열되어 있는 브라우저 내비게이션이나 브라우저 상호 작용 API를 호출할 수 없고 마찬가지로 뒷부분에 나열되어 있는 모든 SWF 간 통신 API를 사용할 수 없습니다.

금지된 API를 호출하면 SecurityError 예외가 발생합니다.

SWF 파일에 대한 참조를 포함하는 HTML 페이지의 <object> 및 <embed> 태그에서 allowNetworking 매개 변수를 설정하려면 다음 예제와 같이 allowNetworking 매개 변수를 추가하고 그 값을 설정합니다.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

또한 HTML 페이지는 스크립트를 사용하여 SWF를 포함하는 태그를 생성할 수 있습니다. 해당 스크립트를 변경하여 올바른 allowNetworking 설정을 삽입해야 합니다. Flash 및 Adobe Flex Builder에서 생성된 HTML 페이지는 AC_FL_RunContent() 함수를 사용하여 SWF 파일에 대한 참조를 포함하므로, 사용자는 다음과 같이 스크립트에 allowNetworking 매개 변수 설정을 추가해야 합니다.

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

다음 API는 allowNetworking이 "internal"로 설정된 경우에는 사용할 수 없습니다.

- navigateToURL()
- fscommand()
- ExternalInterface.call()

이전 목록의 API뿐 아니라 allowNetworking이 "none"으로 설정된 경우 다음 API도 사용할 수 없습니다.

- sendToURL()
- FileReference.download()
- FileReference.upload()
- Loader.load()
- LocalConnection.connect()
- LocalConnection.send()
- NetConnection.connect()
- NetStream.play()
- Security.loadPolicyFile()
- SharedObject.getLocal()
- SharedObject.getRemote()

- `Socket.connect()`
- `Sound.load()`
- `URLLoader.load()`
- `URLStream.load()`
- `XMLSocket.connect()`

선택한 `allowNetworking` 설정이 SWF 파일에서 네트워킹 API를 사용할 수 있도록 허용하는 경우에도 이 장에 설명된 대로 보안 샌드박스 제한을 기반으로 다른 제한 사항이 존재합니다.

`allowNetworking`이 `"none"`으로 설정된 경우에는 TextField 객체의 `htmlText` 속성에 있는 `<img>` 태그에서 외부 미디어를 참조할 수 없습니다(SecurityError 예외 발생).

`allowNetworking`이 `"none"`으로 설정된 경우, 가져와서 Flash 제작 도구(ActionScript가 아님)에 추가한 공유 라이브러리의 심볼이 런타임 시 차단됩니다.

전체 화면 모드 보안

Flash Player 9.0.27.0 이상 버전에서는 Flash 내용으로 전체 화면을 채울 수 있는 전체 화면 모드를 지원합니다. 전체 화면 모드를 사용하려면 Stage의 `displayState` 속성을 `StageDisplayState.FULL_SCREEN` 상수로 설정합니다. 자세한 내용은 367페이지의 "전체 화면 모드 작업"을 참조하십시오.

브라우저에서 실행 중인 SWF 파일에는 몇 가지의 보안 고려 사항이 있습니다.

전체 화면 모드를 사용하려면 다음 예제에 표시된 대로 SWF 파일에 대한 참조를 포함하는 HTML 페이지의 `<object>` 및 `<embed>` 태그에 값이 `"true"`(기본값은 `"false"`)로 설정된 `allowFullScreen` 매개 변수를 추가합니다.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

또한 HTML 페이지는 스크립트를 사용하여 SWF를 포함하는 태그를 생성할 수 있습니다. 해당 스크립트를 변경하여 올바른 `allowFullScreen` 설정을 삽입해야 합니다. Flash 및 Flex Builder에서 생성된 HTML 페이지는 `AC_FL_RunContent()` 함수를 사용하여 SWF 파일에 대한 참조를 포함하므로, 사용자는 다음과 같이 `allowFullScreen` 매개 변수 설정을 추가해야 합니다.

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

전체 화면 모드를 시작하는 ActionScript는 마우스 이벤트나 키보드 이벤트에 대한 응답으로만 호출할 수 있습니다. 기타의 경우에 호출되면 예외가 발생합니다.

전체 화면 모드에서는 텍스트 입력 필드에 텍스트를 입력할 수 없습니다. 전체 화면 모드에서는 모든 키보드 입력 및 키보드 관련 ActionScript가 비활성화됩니다. 이때 응용 프로그램을 일반 모드로 되돌리는 Esc 키 등의 키보드 단축키는 예외입니다.

내용이 전체 화면 모드로 변경되면 사용자에게 종료 방법과 일반 모드로 돌아가는 방법을 알리는 메시지가 나타납니다. 이 메시지는 몇 초 동안 표시된 후 사라집니다.

Stage 객체의 `displayState` 속성을 호출하면 Stage 소유자(기본 SWF 파일)와 다른 보안 샌드박스에 있는 호출자에 대해서는 예외가 발생합니다. 자세한 내용은 730페이지의 “스테이지 보안”을 참조하십시오.

관리자는 mms.cfg 파일에서 `FullScreenDisable = 1`을 설정하여 브라우저에 실행 중인 SWF 파일에 대한 전체 화면 모드를 비활성화할 수 있습니다. 자세한 내용은 710페이지의 “관리자 컨트롤”을 참조하십시오.

브라우저에서 전체 화면 모드를 사용하려면 HTML 페이지에 SWF 파일이 포함되어 있어야 합니다.

독립 실행형 플레이어나 프로젝터 파일에서는 전체 화면 모드가 항상 허용됩니다.

내용 로드

SWF 파일은 다음 유형의 내용을 로드할 수 있습니다.

- SWF 파일
- 이미지
- 사운드
- 비디오

SWF 파일 및 이미지 로드

Loader 클래스를 사용하여 SWF 파일 및 이미지(JPG, GIF, 또는 PNG 파일)를 로드할 수 있습니다. local-with-filesystem 샌드박스에 있는 SWF 파일을 제외한 모든 SWF 파일에서는 모든 네트워크 도메인에 있는 SWF 파일과 이미지를 로드할 수 있습니다. 로컬 샌드박스의 SWF 파일만 로컬 파일 시스템의 SWF 파일과 이미지를 로드할 수 있습니다. 하지만 local-with-networking 샌드박스에 있는 파일은 local-trusted 또는 local-with-networking 샌드박스에 있는 로컬 SWF 파일만 로드할 수 있습니다. local-with-networking 샌드박스에 있는 SWF 파일은 이미지와 같이 SWF 파일이 아닌 로컬 내용을 로드할 수 있지만 로드된 내용의 데이터에는 액세스할 수 없습니다.

신뢰할 수 없는 소스(예: Loader 객체의 루트 SWF 파일에 대한 도메인이 아닌 도메인)에서 SWF 파일을 로드하는 경우, 다음 코드와 같이 Loader 객체의 마스크를 정의하여 해당 마스크의 바깥쪽 Stage 부분에 로드된 내용(Loader 객체의 자식)을 그릴 수 없도록 할 수 있습니다.

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Loader 객체의 `load()` 메서드를 호출하는 경우에는 LoaderContext 객체인 `context` 매개 변수를 지정할 수 있습니다. LoaderContext 클래스에는 로드된 내용을 사용하는 방법에 대한 컨텍스트를 정의할 수 있는 세 가지 속성이 포함되어 있습니다.

- `checkPolicyFile`: 이 속성은 SWF 파일이 아닌 이미지 파일을 로드할 때만 사용합니다. Loader 객체를 포함하는 파일의 도메인을 제외한 도메인의 이미지 파일에 대해 이 속성을 정의합니다. 이 속성을 `true`로 설정하면 Loader는 원래 서버에서 크로스 도메인 정책 파일을 확인합니다(714페이지의 "웹 사이트 컨트롤(크로스 도메인 정책 파일)" 참조). 서버에서 Loader 도메인에 대한 권한을 부여하면 Loader 도메인에 있는 SWF 파일의 ActionScript에서 로드된 이미지의 데이터에 액세스할 수 있습니다. 즉, `Loader.content` 속성을 사용하여 로드된 이미지를 나타내는 Bitmap 객체 또는 로드된 이미지의 픽셀에 액세스하는 `BitmapData.draw()` 메서드에 대한 참조를 얻을 수 있습니다.

- `securityDomain`: 이 속성은 이미지가 아닌 SWF 파일을 로드할 때만 사용합니다. Loader 객체를 포함하는 파일의 도메인과 다른 도메인에서의 SWF 파일에 대해 이 속성을 지정합니다. `securityDomain` 속성의 경우 현재 `null`(기본값) 및 `SecurityDomain.currentDomain`의 두 값만 지원됩니다. `SecurityDomain.currentDomain`을 지정하면 로드된 SWF 파일이 로드하는 SWF 파일의 샌드박스로 *가져와져서* 로드하는 SWF 파일의 자체 서버에서 로드된 것처럼 작동합니다. 이것은 로드된 SWF 파일 서버에 크로스 도메인 정책 파일이 있는 경우, 로드하는 SWF 파일의 도메인에서 액세스할 때만 허용됩니다. 필요한 정책 파일이 있는 경우에는 로드가 시작될 때 로더와 로드되는 파일에서 자유롭게 서로 스크립팅할 수 있는데, 이는 이 두 항목이 동일한 샌드박스에 있기 때문입니다. 대부분의 샌드박스 가져오기 작업은 일반적인 로드를 수행한 다음 로드된 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하는 작업으로 바꿀 수 있습니다. 로드된 SWF 파일이 자체의 고유 샌드박스에 있게 되어 SWF 파일의 실제 서버에 있는 리소스에 액세스할 수 있기 때문에 두 번째 방법이 더욱 사용하기 쉽습니다.
- `applicationDomain`: 이 속성은 ActionScript 3.0으로 작성된 SWF 파일을 로드할 때만 사용합니다. 이미지 파일이나 ActionScript 1.0 또는 2.0으로 작성된 SWF 파일은 해당하지 않습니다. 파일을 로드할 때 사용자는 해당 파일을 로드하는 SWF 파일 응용 프로그램 도메인의 자식인 새 응용 프로그램 도메인(기본 위치)이 아닌 특정 응용 프로그램 도메인에 배치되도록 지정할 수 있습니다. 이때, 응용 프로그램 도메인은 보안 도메인의 하위 개념이며, 따라서 이는 로드하는 SWF 파일을 자체 서버에서 가져왔거나, 또는 `securityDomain` 속성을 사용하여 자체 보안 도메인으로 성공적으로 가져온 경우 즉, 로드하는 SWF 파일이 자체 보안 도메인에 있는 경우에만 대상 응용 프로그램 도메인을 지정할 수 있습니다. 응용 프로그램 도메인을 지정했으나 로드된 SWF 파일이 다른 보안 도메인에 속한 경우에는 `applicationDomain`에 지정된 도메인은 무시됩니다. 자세한 내용은 656페이지의 "ApplicationDomain 클래스 사용"을 참조하십시오.

자세한 내용은 396페이지의 "로드 컨텍스트 지정"을 참조하십시오.

Loader 객체의 중요한 속성은 LoaderInfo 객체인 `contentLoaderInfo` 속성입니다. 다른 대부분의 객체와 달리 LoaderInfo 객체는 로드하는 SWF 파일과 로드된 내용 간에 공유되며 항상 양쪽에서 모두 액세스할 수 있습니다. 로드된 내용이 SWF 파일인 경우 `DisplayObject.loaderInfo` 속성을 통해 LoaderInfo 객체에 액세스할 수 있습니다. LoaderInfo 객체에는 로드 진행률, 로더 및 로드된 내용의 URL, 로더와 로드된 내용 간의 신뢰 관계 및 기타 정보가 포함됩니다. 자세한 내용은 395페이지의 "로드 진행률 모니터링"을 참조하십시오.

사운드 및 비디오 로드

local-with-filesystem에 있는 SWF 파일을 제외한 모든 SWF 파일은 `Sound.load()`, `NetConnection.connect()` 및 `NetStream.play()` 메서드를 사용하여 네트워크에서 사운드와 비디오를 로드할 수 있습니다.

로컬 SWF 파일만 로컬 파일 시스템에서 미디어를 로드할 수 있습니다. local-with-filesystem 샌드박스 또는 local-trusted 샌드박스에 있는 SWF 파일만 이러한 로드된 파일의 데이터에 액세스할 수 있습니다.

로드된 미디어의 데이터에 액세스하는 데는 몇 가지 제한 사항이 있습니다. 자세한 내용은 732페이지의 "데이터로 로드된 미디어 액세스"를 참조하십시오.

텍스트 필드에서 <img> 태그를 사용하여 SWF 파일 및 이미지 로드

다음 코드와 같이 `<img>` 태그를 사용하여 텍스트 필드로 SWF 파일과 비트맵을 로드할 수 있습니다.

```
<img src = 'filename.jpg' id = 'instanceName' >
```

다음 코드와 같이 TextField 인스턴스의 `getImageReference()` 메서드를 사용하여 위와 같은 방식으로 로드된 내용에 액세스할 수 있습니다.

```
var loadedObject:DisplayObject =
  myTextField.getImageReference('instanceName');
```

하지만 이렇게 로드된 SWF 파일과 이미지는 원래 위치에 있는 샌드박스에 배치됩니다.

텍스트 필드에서 `<img>` 태그를 사용하여 이미지를 로드할 때, 이미지의 데이터에 대한 액세스는 크로스 도메인 정책 파일에서 허용됩니다. 다음 코드와 같이 `<img>` 태그에 `checkPolicyFile` 속성을 추가하여 정책 파일을 확인할 수 있습니다.

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

텍스트 필드에서 `<img>` 태그를 사용하여 SWF를 로드할 때 `Security.allowDomain()` 메서드를 호출하여 SWF 파일의 데이터에 대한 액세스를 허용할 수 있습니다.

텍스트 필드의 `<img>` 태그를 사용하여 외부 파일을 로드하는 경우(SWF 파일에 포함된 Bitmap 클래스를 사용하는 경우와는 반대), Loader 객체가 자동으로 TextField 객체의 자식으로 생성되고 ActionScript의 Loader 객체를 사용하여 파일을 로드한 것과 마찬가지로 외부 파일이 해당 Loader에 로드됩니다. 이 경우 `getImageReference()` 메서드에서 자동으로 생성된 Loader를 반환합니다. 호출하는 코드와 동일한 보안 샌드박스에 있기 때문에 이 Loader 객체에 액세스할 때는 별도의 보안 확인이 필요하지 않습니다.

하지만 Loader 객체의 content 속성을 참조하여 로드된 미디어에 액세스하는 경우에는 보안 규칙이 적용됩니다. 내용이 이미지일 경우에는 크로스 도메인 정책 파일을 구현해야 하고, 내용이 SWF 파일인 경우에는 SWF 파일에 allowDomain() 메서드를 호출하는 코드가 있어야 합니다.

RTMP 서버를 사용하여 제공된 내용

Flash Media Server는 RTMP(Real-Time Media Protocol)를 사용하여 데이터, 오디오 및 비디오를 제공합니다. SWF 파일은 RTMP URL을 매개 변수로 전달하고 NetConnection 클래스의 connect() 메서드를 사용하여 이 미디어를 로드합니다. Flash Media Server는 요청하는 파일의 도메인을 기준으로 연결을 제한하고 내용 다운로드를 금지할 수 있습니다. 자세한 내용은 Flash Media Server 설명서를 참조하십시오.

RTMP 소스에서 로드된 미디어의 경우, 런타임 그래픽과 사운드 데이터를 추출하기 위해 BitmapData.draw() 및 SoundMixer.computeSpectrum() 메서드를 사용할 수 없습니다.

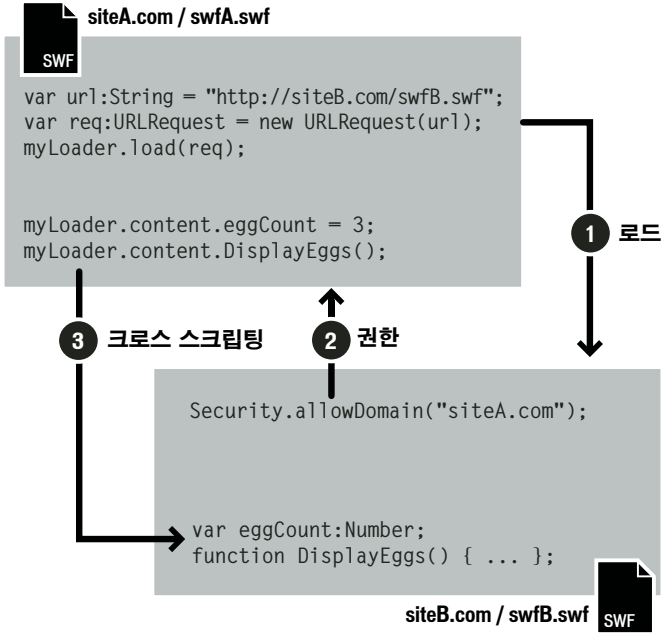
크로스 스크립팅

동일한 도메인에서 ActionScript 3.0으로 작성된 두 개의 SWF 파일을 제공하는 경우에는 (예: 한 SWF 파일의 URL은 http://www.example.com/swfA.swf이고 다른 SWF 파일의 URL은 http://www.example.com/swfB.swf인 경우) 각 SWF 파일에서 다른 SWF 파일의 변수, 객체, 속성, 메서드 및 기타 항목을 검사 및 수정할 수 있습니다. 이를 *크로스 스크립팅*이라고 합니다.

크로스 스크립팅은 AVM1 SWF 파일과 AVM2 SWF 파일 간에는 지원되지 않습니다. AVM1 SWF 파일은 ActionScript 1.0 또는 ActionScript 2.0을 사용하여 작성됩니다. (AVM1 및 AVM2는 ActionScript Virtual Machine을 나타냅니다.) 하지만 LocalConnection 클래스를 사용하여 AVM1과 AVM2 간에 데이터를 보낼 수 있습니다.

ActionScript 3.0으로 작성된 두 SWF 파일이 http://siteA.com/swfA.swf와 http://siteB.com/siteB.swf 같이 서로 다른 도메인에서 제공되는 경우에는 기본적으로 swfA.swf에서 swfB.swf를, swfB.swf에서 swfA.swf를 스크립팅할 수 없습니다. SWF 파일에서는 Security.allowDomain()을 호출하여 다른 도메인의 SWF 파일에 스크립팅 권한을 부여합니다. Security.allowDomain("siteA.com")을 호출하여 swfB.swf는 siteA.com의 SWF 파일에 스크립팅 권한을 부여합니다.

크로스 도메인 상황에서는 관련되는 두 도메인을 명확하게 구별하는 것이 중요합니다. 이 설명서에서는 크로스 스크립팅을 수행하는 쪽을 *액세스하는 항목*(일반적으로 액세스하는 SWF)이라고 하고, 다른 한 쪽을 *액세스되는 항목*(일반적으로 액세스되는 SWF)이라고 합니다. 다음 그림에 표시된 대로 siteA.swf에서 siteB.swf를 스크립팅할 때 siteA.swf는 액세스하는 항목이고 siteB.swf는 액세스되는 항목입니다.



`Security.allowDomain()` 메서드로 설정된 크로스 도메인 권한은 비대칭적입니다. 앞의 예제에서 siteA.swf는 siteB.swf를 스크립팅할 수 있지만 siteB.swf는 siteA.swf를 스크립팅할 수 없습니다. 이는 siteA.swf에서 `Security.allowDomain()` 메서드를 호출하여 siteB.com에 있는 SWF 파일에 스크립팅 권한을 부여하지 않았기 때문입니다. 양쪽 SWF 파일이 `Security.allowDomain()` 메서드를 호출하도록 하여 대칭적 권한을 설정할 수 있습니다.

Flash Player에서 SWF 파일은 다른 SWF 파일의 크로스 도메인 스크립팅으로부터 보호될 뿐 아니라 HTML 파일의 크로스 도메인 스크립팅으로부터도 보호됩니다. HTML에서 SWF로의 스크립팅은 `ExternalInterface.addCallback()` 메서드를 통해 설정된 콜백에서 발생할 수 있습니다. HTML에서 SWF로의 스크립팅이 다른 도메인 간에 발생하는 경우 액세스하는 항목이 SWF 파일일 때와 마찬가지로 액세스되는 SWF 파일에서 `Security.allowDomain()` 메서드를 호출해야 하며, 그렇지 않으면 작업이 실패합니다. 자세한 내용은 717페이지의 "제작자(개발자) 컨트롤"을 참조하십시오.

또한 Flash Player는 SWF에서 HTML로의 스크립팅에 대해 보안 컨트롤을 제공합니다. 자세한 내용은 739페이지의 "호스트 웹 페이지에서 스크립트에 대한 액세스 제어"를 참조하십시오.

스테이지 보안

Stage 객체의 일부 속성 및 메서드를 표시 목록에 있는 모든 sprite 또는 무비 클립에서 사용할 수 있습니다.

그러나 Stage 객체는 첫 번째 SWF 파일이 로드된 경우에 소유자가 있는 것으로 간주됩니다. 기본적으로 다음과 같은 Stage 객체의 속성 및 메서드는 Stage 소유자와 동일한 보안 샌드박스에 있는 SWF 파일에만 사용할 수 있습니다.

속성		메서드
align	showDefaultContextMenu	addChild()
displayState	stageFocusRect	addChildAt()
frameRate	stageHeight	addEventListener()
height	stageWidth	dispatchEvent()
mouseChildren	tabChildren	hasEventListener()
numChildren	textSnapshot	setChildIndex()
quality	width	willTrigger()
scaleMode		

Stage 소유자의 샌드박스가 아닌 샌드박스에 있는 SWF 파일에서 위의 속성과 메서드에 액세스하려면 Stage 소유자 SWF 파일에서 Security.allowDomain() 메서드를 호출하여 외부 샌드박스의 도메인을 허용해야 합니다. 자세한 내용은 717페이지의 “제작자(개발자) 컨트롤”을 참조하십시오.

frameRate 속성은 특별한 경우로 모든 SWF 파일에서 frameRate 속성을 읽을 수 있습니다. 그러나 Stage 소유자의 보안 샌드박스에 있거나 Security.allowDomain() 메서드를 호출하여 권한을 부여한 SWF 파일에서만 이 속성을 변경할 수 있습니다.

Stage 객체의 removeChildAt() 및 swapChildrenAt() 메서드에도 제한 사항이 있지만 다른 제한 사항과는 다릅니다. 이 메서드를 호출하려면 코드가 Stage 소유자와 동일한 도메인이 아니라 영향 받는 자식 객체 또는 Security.allowDomain() 메서드를 호출할 수 있는 자식 객체의 소유자와 동일한 도메인에 있어야 합니다.

표시 목록 탐색

다른 샌드박스에서 로드된 표시 객체에 액세스하는 SWF 파일의 기능은 제한적입니다. SWF 파일에서 다른 샌드박스에 있는 다른 SWF 파일에서 생성된 표시 객체에 액세스하려면 액세스되는 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하여 액세스하는 SWF 파일의 도메인에 액세스를 허용해야 합니다. 자세한 내용은 717페이지의 "제작자(개발자) 컨트롤"을 참조하십시오.

Loader 객체에서 로드한 Bitmap 객체에 액세스하려면 해당 이미지 파일의 원래 서버에 크로스 도메인 정책 파일이 있어야 하고 해당 크로스 도메인 정책 파일에서 Bitmap 객체에 액세스를 시도하는 SWF 파일의 도메인에 액세스 권한을 부여해야 합니다(714페이지의 "웹 사이트 컨트롤(크로스 도메인 정책 파일)" 참조).

로드된 파일 및 Loader 객체에 해당하는 LoaderInfo 객체에는 로드된 객체와 Loader 객체 간의 관계를 정의하는 `childAllowsParent`, `parentAllowsChild` 및 `sameDomain`의 세 가지 속성이 있습니다.

이벤트 보안

표시 목록과 연관된 이벤트에는 해당 이벤트를 전달하는 표시 객체의 샌드박스를 기준으로 보안 액세스 제한이 적용됩니다. 표시 목록의 이벤트에는 버블링과 캡처 단계가 있습니다(295페이지의 제10장, "이벤트 처리" 참조). 버블링 및 캡처 단계에서 이벤트는 소스 표시 객체에서 표시 목록의 부모 표시 객체를 통해 마이그레이션됩니다. 부모 객체가 소스 표시 객체와 다른 보안 샌드박스에 있는 경우에는 부모 객체 소유자와 소스 객체 소유자 간에 상호 신뢰가 없는 한 해당 부모 객체 아래에서 캡처 및 버블링 단계가 중단됩니다. 상호 신뢰는 다음 작업을 통해 얻을 수 있습니다.

1. 부모 객체를 소유한 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하여 소스 객체를 소유한 SWF 파일의 도메인을 신뢰해야 합니다.
2. 소스 객체를 소유한 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하여 부모 객체를 소유한 SWF 파일의 도메인을 신뢰해야 합니다.

로드된 파일 및 Loader 객체에 해당하는 LoaderInfo 객체에는 로드된 객체와 Loader 객체 간의 관계를 정의하는 `childAllowsParent` 및 `parentAllowsChild`의 두 가지 속성이 있습니다.

표시 객체 이외의 객체에서 전달된 이벤트에 대해서는 보안 확인이나 보안 관련 지정이 없습니다.

데이터로 로드된 미디어 액세스

BitmapData.draw() 및 SoundMixer.computeSpectrum()과 같은 메서드를 사용하여 로드된 데이터에 액세스합니다. 기본적으로 한 보안 샌드박스의 SWF 파일은 다른 보안 샌드박스에 있는 로드된 미디어로 렌더링되거나 재생되는 그래픽이나 오디오 객체에서 픽셀 데이터나 오디오 데이터를 가져올 수 없습니다. 하지만 다음 메서드를 사용하여 이러한 권한을 부여할 수 있습니다.

- 로드된 SWF 파일에서 Security.allowDomain() 메서드를 호출하여 다른 도메인의 SWF 에 대한 데이터 액세스 권한을 부여합니다.
- 로드된 이미지, 사운드 또는 비디오의 경우 로드된 파일의 서버에 크로스 도메인 정책 파일을 추가합니다. 이 정책 파일에서 BitmapData.draw() 또는 SoundMixer.computeSpectrum() 메서드 호출을 시도하는 SWF 파일의 도메인에 대한 액세스 권한을 부여해야 해당 파일에서 데이터를 추출할 수 있습니다.

다음 단원에서는 비트맵, 사운드 및 비디오 데이터 액세스에 대해 자세히 설명합니다.

비트맵 데이터 액세스

BitmapData 객체의 draw() 메서드를 사용하면 BitmapData 객체에 현재 표시되는 모든 표시 객체의 픽셀을 그릴 수 있습니다. 여기에는 MovieClip 객체, Bitmap 객체 또는 모든 표시 객체의 픽셀이 포함됩니다. 다음 조건이 충족되어야 draw() 메서드로 BitmapData 객체에 픽셀을 그릴 수 있습니다.

- 로드된 비트맵이 아닌 소스 객체의 경우, 소스 객체와 (Sprite 또는 MovieClip 객체의 경우) 모든 자식 객체가 draw() 메서드를 호출하는 객체와 동일한 도메인에 있거나 또는 Security.allowDomain() 메서드를 호출하여 해당 호출자에서 액세스할 수 있는 SWF 파일에 있어야 합니다.
- 로드된 비트맵 소스 객체의 경우에는 소스 객체가 draw() 메서드를 호출하는 객체와 동일한 도메인에 있거나 또는 해당 소스 서버에 호출하는 도메인에 권한을 부여하는 크로스 도메인 정책 파일이 포함되어 있어야 합니다.

이러한 조건이 충족되지 않으면 SecurityError 예외가 발생합니다.

Loader 클래스의 load() 메서드를 사용하여 이미지를 로드하는 경우 LoaderContext 객체인 context 매개 변수를 지정할 수 있습니다. LoaderContext 객체의 checkPolicyFile 속성을 true로 설정하면 Flash Player에서 이미지가 로드되는 서버의 크로스 도메인 정책 파일을 확인합니다. 크로스 도메인 정책 파일이 있고 이 파일에서 로드하는 SWF 파일의 도메인을 허용할 경우, 해당 SWF 파일은 Bitmap 객체의 데이터에 액세스할 수 있고, 그렇지 않으면 액세스할 수 없습니다.

또한 텍스트 필드의 <img> 태그를 통해 로드된 이미지의 checkPolicyFile 속성을 지정할 수 있습니다. 자세한 내용은 727페이지의 “텍스트 필드에서 <img> 태그를 사용하여 SWF 파일 및 이미지 로드”를 참조하십시오.

사운드 데이터 액세스

다음과 같은 사운드 관련 ActionScript 3.0 API에는 보안 제한 사항이 있습니다.

- SoundMixer.computeSpectrum() 메서드 - 사운드 파일과 동일한 보안 샌드박스에 있는 SWF 파일을 항상 허용합니다. 다른 샌드박스의 파일에 대해서는 보안 확인을 실행합니다.
- SoundMixer.stopAll() 메서드 - 사운드 파일과 동일한 보안 샌드박스에 있는 SWF 파일을 항상 허용합니다. 다른 샌드박스의 파일에 대해서는 보안 확인을 실행합니다.
- Sound 클래스의 id3 속성 - 사운드 파일과 동일한 보안 샌드박스에 있는 SWF 파일을 항상 허용합니다. 다른 샌드박스의 파일에 대해서는 보안 확인을 실행합니다.

모든 사운드에는 내용 샌드박스와 소유자 샌드박스의 두 가지 관련 샌드박스가 있습니다.

- 사운드의 원래 도메인에 의해 내용 샌드박스가 결정되며 여기에서 사운드의 id3 속성과 SoundMixer.computeSpectrum() 메서드를 통해 사운드에서 데이터를 추출할 수 있는지 여부가 결정됩니다.
- 사운드 재생을 시작한 객체에 의해 소유자 샌드박스가 결정되며 여기에서 SoundMixer.stopAll() 메서드를 사용하여 사운드를 중단할 수 있는지 여부가 결정됩니다.

Sound 클래스의 load() 메서드를 사용하여 사운드를 로드하는 경우 SoundLoaderContext 객체인 context 매개 변수를 지정할 수 있습니다. SoundLoaderContext 객체의 checkPolicyFile 속성을 true로 설정하면 Flash Player에서 사운드가 로드되는 서버의 크로스 도메인 정책 파일을 확인합니다. 크로스 도메인 정책 파일이 있고 이 파일이 로드하는 SWF 파일의 도메인을 허용할 경우, 해당 SWF 파일은 Sound 객체의 id 속성에 액세스할 수 있고, 그렇지 않으면 액세스할 수 없습니다. 또한 checkPolicyFile 속성을 설정하여 로드된 사운드에 SoundMixer.computeSpectrum() 메서드를 사용할 수 있습니다.

하나 이상의 사운드 소유자 샌드박스를 호출자에서 액세스할 수 없어 SoundMixer.stopAll() 메서드 호출 시 모든 사운드가 중단되지 않는지 여부를 SoundMixer.areSoundsInaccessible() 메서드를 사용하여 확인할 수 있습니다.

SoundMixer.stopAll() 메서드를 호출하면 stopAll()의 호출자와 동일한 소유자 샌드박스에 있는 사운드를 중단할 수 있습니다. 또한 Security.allowDomain() 메서드를 호출한 SWF 파일에서 재생을 시작한 사운드를 중단하여 stopAll() 메서드를 호출하는 SWF 파일의 도메인에 액세스를 허용할 수 있습니다. 다른 사운드는 중단되지 않으며, 이 사운드는 SoundMixer.areSoundsInaccessible() 메서드를 호출하여 확인할 수 있습니다.

computeSpectrum() 메서드를 호출하려면 재생되는 모든 사운드가 해당 메서드를 호출하는 객체와 동일한 샌드박스에 있거나 또는 호출자의 샌드박스에 액세스 권한을 부여한 소스에 있어야 합니다. 그렇지 않으면 SecurityError 예외가 발생합니다. SWF 파일의 라이브러리에 포함된 사운드에서 로드된 사운드의 경우, 로드된 SWF 파일에서 Security.allowDomain() 메서드를 호출하면 권한이 부여됩니다. SWF 파일이 아닌 소스에서 로드된 사운드의 경우(로드된 mp3 파일 또는 Flash 비디오에서 시작), 소스 서버의 크로스 도메인 정책 파일에서 로드된 미디어의 데이터에 대한 액세스 권한을 부여합니다. 사운드를 RTMP 스트림에서 로드한 경우에는 computeSpectrum() 메서드를 사용할 수 없습니다.

자세한 내용은 717페이지의 “제작자(개발자) 컨트롤” 및 714페이지의 “웹 사이트 컨트롤(크로스 도메인 정책 파일)”을 참조하십시오.

비디오 데이터 액세스

BitmapData.draw() 메서드를 사용하여 현재 비디오 프레임의 픽셀 데이터를 캡처할 수 있습니다.

다음과 같은 두 종류의 비디오가 있습니다.

- RTMP 비디오
- RTMP 서버 없이 FLV 파일에서 로드되는 점진적 비디오

BitmapData.draw() 메서드를 사용하여 RTMP 비디오에 액세스할 수 없습니다.

점진적 비디오를 source 매개 변수로 하여 BitmapData.draw() 메서드를 호출하는 경우, BitmapData.draw()의 호출자가 FLV 파일과 동일한 샌드박스에 있거나, 또는 FLV 파일의 서버에 호출하는 SWF 파일의 도메인에 대해 권한을 부여하는 정책 파일을 있어야 합니다. NetStream 객체의 checkPolicyFile 속성을 true로 설정하여 해당 정책 파일의 다운로드를 요청할 수 있습니다.

데이터 로드

SWF 파일은 서버에서 ActionScript로 데이터를 로드하고 ActionScript에서 서버로 데이터를 보낼 수 있습니다. 로드된 정보가 미디어로 표시되지 않고 ActionScript에 직접 표시되므로, 데이터 로드 작업은 미디어 로드 작업과는 다릅니다. 일반적으로 SWF 파일은 자체 도메인에서 데이터를 로드할 수 있습니다. 그러나 다른 도메인에서 데이터를 로드하려면 보통 크로스 도메인 정책 파일이 필요합니다.

URLLoader 및 URLStream 사용

XML 파일이나 텍스트 파일과 같은 데이터를 로드할 수 있습니다. URLLoader 및 URLStream 클래스의 `load()` 메서드는 크로스 도메인 정책 파일 권한으로 제어합니다.

`load()` 메서드를 사용하여 해당 메서드를 호출하는 SWF 파일의 도메인이 아닌 다른 도메인의 내용을 로드하는 경우, Flash Player는 로드되는 에셋의 서버에서 크로스 도메인 정책 파일을 확인합니다. 크로스 도메인 정책 파일이 있으면 로드하는 SWF 파일의 도메인에 대한 액세스가 허용되고 데이터를 로드할 수 있습니다.

소켓 연결

소켓 및 XML 소켓 연결에 대한 크로스 도메인 액세스는 기본적으로 비활성화됩니다. 또한 1024 미만의 포트에서 SWF 파일과 동일한 도메인에 있는 소켓 연결에 대한 액세스도 기본적으로 비활성화됩니다. 사용자는 다음 위치에서 크로스 도메인 정책 파일을 제공하여 이러한 포트에 대한 액세스를 허용할 수 있습니다.

- 기본 소켓 연결과 동일한 포트
- 다른 포트
- 소켓 서버와 동일한 도메인의 포트 80에 있는 HTTP 서버

기본 소켓 연결과 동일한 포트 또는 다른 포트에서 크로스 도메인 정책 파일을 제공하는 경우, 다음 예제와 같이 크로스 도메인 정책 파일의 `to-ports` 속성을 사용하여 허용되는 포트를 열거할 수 있습니다.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example.org" to-ports="516-523" />
   <allow-access-from domain="adobe.com" to-ports="507,516-523" />
   <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

기본 소켓 연결과 동일한 포트에서 소켓 정책 파일을 가져오려면 단순히 `Socket.connect()` 또는 `XMLSocket.connect()` 메서드를 호출합니다. 지정된 도메인이 호출하는 SWF 파일의 도메인과 다른 경우 Flash Player에서는 자동으로 사용자가 연결하려는 기본 연결과 동일한 포트에서 정책 파일을 가져오려고 시도합니다. 기본 연결과 동일한 서버의 다른 포트에서 소켓 정책 파일을 가져오려면, 다음과 같이 특수 `"xmlsocket"` 구문을 사용하여 `Security.loadPolicyFile()` 메서드를 호출합니다.

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Security.loadPolicyFile() 메서드를 Socket.connect() 또는 XMLSocket.connect() 메서드보다 먼저 호출합니다. 그러면 Flash Player은 정책 파일 요청이 수행될 때까지 기다렸다가 기본 연결을 허용할지 여부를 결정합니다.

소켓 서버를 구현하고 소켓 정책 파일을 제공해야 할 필요가 있는 경우에는 기본 연결을 허용하는 포트와 동일한 포트를 사용하여 정책 파일을 제공할지 아니면 다른 포트에서 정책 파일을 제공할지 여부를 결정합니다. 두 경우 모두 서버는 클라이언트에서 첫 번째 전송을 수행할 때까지 기다렸다가 정책 파일을 보낼지 또는 기본 연결을 설정할지 여부를 결정합니다. Flash Player에서 정책 파일을 요청할 때는 항상 연결이 설정된 직후에 다음 문자열을 전송합니다.

```
<policy-file-request/>
```

서버는 이 문자열을 수신한 후 정책 파일을 전송할 수 있습니다. 정책 파일 요청과 기본 연결 모두에 동일한 연결을 다시 사용하지 마십시오. 정책 파일을 전송한 후에는 해당 연결을 닫아야 합니다. 연결을 닫지 않으면, Flash Player에서 기본 연결을 설정하기 위해 다시 연결하기 전에 정책 파일 연결을 닫습니다.

자세한 내용은 716페이지의 “소켓 정책 파일”을 참조하십시오.

데이터 보내기

데이터 전송은 SWF 파일의 ActionScript 코드에서 서버 또는 리소스에 데이터를 보낼 때 발생합니다. 네트워크 도메인 SWF 파일에는 데이터 전송이 항상 허용됩니다. 로컬 SWF 파일은 local-trusted 또는 local-with-networking 샌드박스에 있는 경우에만 네트워크 주소로 데이터를 보낼 수 있습니다. 자세한 내용은 719페이지의 “로컬 샌드박스”를 참조하십시오.

flash.net.sendToURL() 함수를 사용하여 URL로 데이터를 보낼 수 있습니다. 기타 메서드를 사용하여 URL로 요청을 보낼 수도 있습니다. 여기에는 Loader.load() 및 Sound.load()와 같은 로드 메서드, URLLoader.load() 및 URLStream.load()와 같은 데이터 로드 메서드가 포함됩니다.

파일 업로드 및 다운로드

`FileReference.upload()` 메서드는 원격 서버로 사용자가 선택한 파일의 업로드를 시작합니다. `FileReference.browse()` 또는 `FileReferenceList.browse()` 메서드는 `FileReference.upload()` 메서드보다 먼저 호출해야 합니다.

`FileReference.download()` 메서드를 호출하면 원격 서버에서 파일을 다운로드할 수 있는 대화 상자가 열립니다.

예외	서버에 사용자 인증이 필요한 경우 브라우저에서 실행 중인, 즉 브라우저 플러그 인이나 ActiveX 컨트롤을 사용하는 SWF 파일만이 인증 및 다운로드에 필요한 사용자 이름과 암호를 요청하는 대화 상자를 제공할 수 있습니다. Flash Player는 사용자 인증을 요청하는 서버에 대한 업로드를 허용하지 않습니다.

호출하는 SWF 파일이 local-with-filesystem 샌드박스에 있는 경우에는 업로드 및 다운로드가 허용되지 않습니다.

기본적으로 SWF 파일은 자체 서버가 아닌 서버에서의 업로드나 다운로드를 시작하지 않습니다. 호출하는 SWF 파일의 도메인에 권한을 부여하는 크로스 도메인 정책 파일이 서버에서 제공되는 경우, SWF 파일은 다른 서버로부터의 업로드 및 다운로드를 허용합니다.

보안 도메인으로 가져온 SWF 파일에서 포함된 내용 로드

SWF 파일을 로드하는 경우 해당 파일을 로드하는 데 사용되는 Loader 객체의 `load()` 메서드에 대한 `context` 매개 변수를 설정할 수 있습니다. 이 매개 변수는 LoaderContext 객체를 사용합니다. LoaderContext 객체의 `securityDomain` 속성을 `Security.currentDomain`으로 설정하면 Flash Player에서 로드된 SWF 파일의 서버에서 크로스 도메인 정책 파일을 확인합니다. 크로스 도메인 정책 파일이 있으면 로드하는 SWF 파일의 도메인에 대한 액세스가 허용되고 SWF 파일을 가져온 미디어로 로드할 수 있습니다. 이렇게 하여 로드하는 파일이 SWF 파일의 라이브러리에 있는 객체에 액세스할 수 있습니다.

SWF 파일에서 다른 보안 샌드박스에 로드된 SWF 파일의 클래스에 액세스하는 다른 방법은, 로드된 SWF 파일에서 `Security.allowDomain()` 메서드를 호출하여 호출하는 SWF 파일의 도메인에 대한 액세스를 허용하는 것입니다. 로드된 SWF 파일의 기본 클래스 생성자 메서드에 `Security.allowDomain()` 메서드에 대한 호출을 추가한 다음, 로드하는 SWF 파일에서 Loader 객체의 `contentLoaderInfo` 속성으로부터 전달되는 `init` 이벤트에 응답하는 이벤트 리스너를 추가하도록 할 수 있습니다. 이 이벤트가 전달되면 생성자 메서드에서 `Security.allowDomain()` 메서드를 호출했던 로드된 SWF 파일과 로드된 SWF 파일의 클래스를 로드하는 SWF 파일에서 사용할 수 있습니다. 로드하는 SWF 파일은 `Loader.contentLoaderInfo.applicationDomain.getDefinition()`을 호출하여 로드된 SWF 파일에서 클래스를 가져올 수 있습니다.

이전 내용으로 작업

Flash Player 6에서 특정 Flash Player 설정에 사용되는 도메인은 SWF 파일 도메인의 뒷부분을 기준으로 합니다. 이러한 설정에는 카메라 및 마이크 권한, 저장소 할당량, 영구 공유 객체 저장소 설정이 포함됩니다.

예를 들어, SWF 파일의 도메인이 www.example.com과 같이 세 부분 이상으로 구성된 경우 도메인의 첫 번째 부분(www)이 제거되고 도메인의 나머지 부분이 사용됩니다. 따라서 Flash Player 6에서는 www.example.com과 store.example.com 모두에서 설정에 example.com 도메인을 사용합니다. 마찬가지로 www.example.co.uk와 store.example.co.uk 모두에서 설정에 example.co.uk 도메인을 사용합니다. 이렇게 하면 example1.co.uk 및 example2.co.uk와 같은 관련되지 않은 도메인의 SWF 파일에서 동일한 공유 객체에 액세스하는 문제가 발생할 수 있습니다.

Flash Player 7 이상 버전에서는 기본적으로 SWF 파일의 정확한 도메인에 따라 플레이어 설정이 선택됩니다. 예를 들어, www.example.com의 SWF 파일은 www.example.com의 플레이어 설정을 사용하고, store.example.com의 SWF 파일은 store.example.com에 별도로 설정된 플레이어 설정을 사용합니다.

ActionScript 3.0을 사용하여 작성된 SWF 파일에서 `Security.exactSettings`가 `true`(기본값)로 설정된 경우 Flash Player는 플레이어 설정에 정확히 일치하는 도메인을 사용합니다. `false`로 설정된 경우에는 Flash Player 6에서 사용되는 도메인 설정을 사용합니다. 설정 기본값에서 `exactSettings`를 변경하려면 Flash Player에서 카메라 또는 마이크를 사용하거나 영구 공유 객체를 가져오는 등 플레이어 설정을 선택하는 이벤트가 발생하기 전에 변경해야 합니다.

버전 6 SWF 파일을 제작하고 여기에서 영구 공유 객체를 작성한 경우, ActionScript 3.0을 사용하는 SWF에서 영구 공유 객체를 가져오려면 `SharedObject.getLocal()`을 호출하기 전에 `Security.exactSettings`를 `false`로 설정해야 합니다.

LocalConnection 권한 설정

LocalConnection 클래스를 사용하여 서로에게 지시 사항을 보낼 수 있는 SWF 파일을 개발할 수 있습니다. LocalConnection 객체는 동일한 클라이언트 컴퓨터에서 실행되고 있는 SWF 파일 간에서만 통신할 수 있습니다. 그러나 이 SWF 파일들은 서로 다른 응용 프로그램에서 실행될 수 있습니다. 예를 들어, 한 SWF 파일은 브라우저에서, 다른 SWF 파일은 프로젝터에서 실행될 수 있습니다.

모든 LocalConnection 통신에는 센더 SWF 파일과 리스너 SWF 파일이 있습니다. 기본적으로 Flash Player는 동일한 도메인에 있는 SWF 파일 간에 LocalConnection 통신을 허용합니다. 서로 다른 샌드박스에 있는 SWF 파일의 경우에는 리스너에서 `LocalConnection.allowDomain()` 메서드를 사용하여 센더 권한을 허용해야 합니다. `LocalConnection.allowDomain()` 메서드에 인수로 전달하는 문자열에는 정확한 도메인 이름, IP 주소 및 * 와일드카드를 포함할 수 있습니다.

ActionScript 3.0

`allowDomain()` 메서드는 ActionScript 1.0 및 2.0에서의 형식과는 다르게 변경되었습니다. 이전 버전에서 `allowDomain()`은 구현되는 콜백 메서드였습니다. ActionScript 3.0에서 `allowDomain()`은 호출되는 LocalConnection 클래스의 내장 메서드입니다. 이러한 변경으로 인해 `allowDomain()`은 `Security.allowDomain()`과 거의 동일한 방식으로 작동됩니다.

SWF 파일은 LocalConnection 클래스의 `domain` 속성을 사용하여 도메인을 확인합니다.

호스트 웹 페이지에서 스크립트에 대한 액세스 제어

아웃바운드 스크립팅은 다음과 같은 ActionScript 3.0 API를 사용하여 수행됩니다.

- `flash.system.fscommand()` 함수
- `flash.net.navigateToURL()` 함수(`navigateToURL("javascript: alert('Hello from Flash Player.')"` 등의 스크립팅 문을 지정하는 경우)
- `flash.net.navigateToURL()` 함수(`window` 매개 변수가 `"_top"`, `"_self"` 또는 `"_parent"`로 설정된 경우)
- `ExternalInterface.call()` 메서드

로컬로 실행 중인 SWF 파일의 경우 SWF 파일과 포함하는 웹 페이지(있는 경우)가 local-trusted 보안 샌드박스에 있는 경우에만 이러한 메서드를 성공적으로 호출할 수 있습니다. 내용이 local-with-networking 또는 local-with-filesystem 샌드박스에 있는 경우에는 해당 메서드에 대한 호출이 실패합니다.

SWF 파일을 로드하는 HTML 코드의 `AllowScriptAccess` 매개 변수는 SWF 파일 내에서 아웃바운드 스크립팅을 수행하는 기능을 제어합니다.

SWF 파일을 호스트하는 웹 페이지의 HTML 코드에 이 매개 변수를 설정합니다. PARAM 또는 EMBED 태그에 매개 변수를 설정합니다.

AllowScriptAccess 매개 변수는 가능한 세 가지 값, "always", "sameDomain" 또는 "never" 중 하나를 가질 수 있습니다.

- AllowScriptAccess가 "sameDomain"이면 SWF 파일과 해당 웹 페이지가 동일한 도메인에 있는 경우에만 아웃바운드 스크립팅이 허용됩니다. AVM2 내용의 기본값입니다.
- AllowScriptAccess가 "never"이면 아웃바운드 스크립팅은 항상 실패합니다.
- AllowScriptAccess가 "always"이면 아웃바운드 스크립팅은 항상 성공합니다.

AllowScriptAccess 매개 변수가 HTML 페이지의 SWF 파일에 지정되지 않은 경우에는 AVM2 내용에는 "sameDomain"이 기본값으로 사용됩니다.

다음은 HTML 페이지에 AllowScriptAccess 태그를 설정하는 예입니다.

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-
  444553540000' codebase='http://download.adobe.com/pub/shockwave/cabs/
  flash/swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='AllowScriptAccess' value='never'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/
  getflashplayer' src='MyMovie.swf' height='100%' width='100%'
  AllowScriptAccess='never'/>
</object>
```

AllowScriptAccess 매개 변수는 한 도메인에서 호스팅되는 SWF 파일이 다른 도메인의 HTML 페이지에 있는 스크립트에 액세스하지 못하도록 할 수 있습니다. 다른 도메인에서 호스팅되는 모든 SWF 파일에 AllowScriptAccess="never"를 사용하여 HTML 페이지에 있는 스크립트의 보안을 유지할 수 있습니다.

자세한 내용은 *ActionScript 3.0 언어 및 구성 요소 참조 설명서*에서 다음 항목을 참조하십시오.

- flash.system.fscommand() 함수
- flash.net.navigateToURL() 함수
- ExternalInterface 클래스의 call() 메서드

공유 객체

Flash Player는 *공유 객체*를 사용하는 기능을 제공합니다. 공유 객체는 SWF 파일의 외부에 즉, 사용자의 파일 시스템에 로컬로 또는 RTMP 서버에 원격으로 존속하는 ActionScript 객체입니다. 공유 객체는 Flash Player의 다른 미디어와 같이 보안 샌드박스로 구분됩니다. 하지만 공유 객체는 도메인 경계를 넘어 액세스할 수 있는 리소스가 아니므로 공유 객체에 대한 샌드박스 모델은 약간 다릅니다. 대신 공유 객체는 항상 SharedObject 클래스의 메서드를 호출하는 각 SWF 파일의 도메인에 대한 특정 공유 객체 저장소에서 가져옵니다. 보통 공유 객체 저장소는 SWF 파일의 도메인보다 세부적으로 구분되어 있으며, 기본적으로 각 SWF 파일은 전체 원래 URL에 대한 특정 공유 객체 저장소를 사용합니다.

SWF 파일은 `SharedObject.getLocal()` 및 `SharedObject.getRemote()` 메서드의 `localPath` 매개 변수를 사용하여 해당 URL의 일부와 연관된 공유 객체 저장소를 사용할 수 있습니다. 이렇게 하여 SWF 파일은 다른 URL의 다른 SWF 파일과 공유 객체 저장소를 공유할 수 있습니다. `localPath` 매개 변수로 `'/'`를 전달하는 경우에도 해당 도메인에 대한 특정 공유 객체 저장소가 지정됩니다.

사용자는 [Flash Player 설정] 대화 상자 또는 설정 관리자를 사용하여 공유 객체 액세스를 제한할 수 있습니다. 기본적으로 도메인당 최대 100KB의 공유 객체 데이터를 만들 수 있습니다. 관리자와 사용자는 또한 파일 시스템에 대한 쓰기 권한을 제한할 수도 있습니다. 자세한 내용은 710페이지의 “관리자 컨트롤” 및 712페이지의 “사용자 컨트롤”을 참조하십시오.

`SharedObject.getLocal()` 메서드 또는 `SharedObject.getRemote()` 메서드의 `secure` 매개 변수에 `true`를 지정하여 보안 공유 객체를 지정할 수 있습니다. `secure` 매개 변수의 다음 사항에 주의하십시오.

- 이 매개 변수를 `true`로 설정하면 Flash Player가 보안 공유 객체를 새로 만들거나 기존 보안 공유 객체에 대한 참조를 가져옵니다. 이 보안 공유 객체는 `secure` 매개 변수가 `true`로 설정된 `SharedObject.getLocal()`을 호출하는 HTTPS를 통해 제공되는 SWF 파일에서만 읽거나 쓸 수 있습니다.
- 이 매개 변수가 `false`로 설정되면 Flash Player는 비 HTTPS 연결을 통해 제공되는 SWF 파일에서 읽거나 쓸 수 있는 새 공유 객체를 만들거나 기존 공유 객체에 대한 참조를 가져옵니다.

호출하는 SWF 파일이 HTTPS URL에 없는 경우, `SharedObject.getLocal()` 메서드 또는 `SharedObject.getRemote()` 메서드의 `secure` 매개 변수를 `true`로 지정하면 SecurityError 예외가 발생합니다.

공유 객체 저장소의 선택은 SWF 파일의 원래 URL을 기반으로 합니다. 이것은 SWF 파일이 단순 URL에서 시작되지 않은 경우 즉, 가져오기 로드 및 동적 로드의 두 상황에도 적용됩니다. 가져오기 로드는 `LoaderContext.securityDomain` 속성이 `SecurityDomain.currentDomain`으로 설정된 SWF 파일을 로드하는 경우입니다. 이 경우 로드된 SWF 파일은 로드하는 SWF 파일의 도메인에서 시작하여 실제 원래 URL을 지정하는 의사 URL을 가집니다. 동적 로드는 `Loader.loadBytes()` 메서드를 사용하여 SWF 파일을 로드하는 것을 말합니다. 이 경우 로드된 SWF 파일은 로드하는 SWF 파일의 전체 URL 다음에 정수 ID가 오는 의사 URL을 가집니다. 가져오기 로드 및 동적 로드의 두 경우 모두 `LoaderInfo.url` 속성을 사용하여 SWF 파일의 의사 URL을 검사할 수 있습니다. 의사 URL은 공유 객체 저장소 선택 시 실제 URL과 동일하게 취급됩니다. 의사 URL의 일부 또는 전체를 사용하는 공유 객체 `localPath` 매개 변수를 지정할 수 있습니다.

사용자 및 관리자는 *타사 공유 객체*를 사용하지 못하도록 지정할 수 있습니다. SWF 파일의 원래 URL이 브라우저의 주소 표시줄에 표시된 URL과 다른 도메인인 경우에는 웹 브라우저에서 실행 중인 모든 SWF 파일에서 공유 객체를 사용합니다. 사용자 및 관리자는 크로스 도메인 추적을 방지하고 개인 정보를 보호하기 위해 타사 공유 객체를 사용하지 못하도록 지정할 수 있습니다. 이 제한을 피하려면 SWF 파일이 브라우저의 주소 표시줄에 표시된 도메인과 동일한 도메인에서 제공되도록 하는 HTML 페이지 구조 내에서만 공유 객체를 사용하는 SWF 파일이 로드되도록 해야 합니다. 타사 SWF 파일의 공유 객체를 사용하려고 시도하면 타사 공유 객체 사용이 비활성화되고 `SharedObject.getLocal()` 및 `SharedObject.getRemote()` 메서드에서 `null`을 반환합니다. 자세한 내용은 www.adobe.com/products/flashplayer/articles/thirdpartylso를 참조하십시오.

카메라, 마이크, 클립보드, 마우스 및 키보드 액세스

SWF 파일에서 `Camera.get()` 또는 `Microphone.get()` 메서드를 사용하여 사용자의 카메라나 마이크에 액세스하려고 시도하면 Flash Player에서 사용자가 카메라와 마이크에 대한 액세스를 허용하거나 거부할 수 있는 [개인 정보] 대화 상자를 표시합니다. 또한 사용자와 관리자는 mms.cfg 파일의 컨트롤, 설정 UI 및 설정 관리자를 통해 사이트별로 또는 전체 사이트에 대한 카메라 액세스를 비활성화할 수 있습니다(710페이지의 "관리자 컨트롤" 및 712페이지의 "사용자 컨트롤" 참조). 사용자 제한이 있는 `Camera.get()` 및 `Microphone.get()` 메서드 각각은 `null` 값을 반환합니다. `Capabilities.avHardwareDisable` 속성을 사용하여 카메라 및 마이크가 관리상의 이유로 금지되었는지(`true`) 또는 허용되었는지(`false`) 여부를 확인할 수 있습니다.

`System.setClipboard()` 메서드를 사용하면 SWF 파일에서 클립보드의 내용을 일반 텍스트 문자열로 바꿀 수 있습니다. 이로 인한 보안상의 위험은 없습니다. 암호 및 기타 중요한 데이터를 클립보드로 잘라내거나 복사하는 경우 이에 따른 위험을 방지하기 위해, 클립보드의 내용을 읽을 수 있는 "getClipboard"(read) 메서드는 제공되지 않습니다.

Flash 응용 프로그램은 자체 포커스 안에서 발생하는 키보드와 마우스 이벤트만 모니터링할 수 있고 다른 응용 프로그램의 키보드 또는 마우스 이벤트는 감지할 수 없습니다.

색 인

가

나

다

라

마

바

사

아

자

차

카

타

파

하

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

기호

숫자