Utvecklarhandbok för ACTIONSCRIPT® 3.0

Juridiska meddelanden

Mer information om juridiska meddelanden finns i http://help.adobe.com/sv_SE/legalnotices/index.html.

Senast uppdaterad 2014-12-2

Innehåll

Kapitel 1: Arbeta med datum och tidpunkter
Hantera kalenderdatum och tidpunkter 1
Kontrollera tidsintervall 4
Exempel på datum och tid: enkel analog klocka 6

Kapitel 2: Arbeta med strängar
Hur du arbetar med strängar 10
Skapa strängar 11
Egenskapen Length 12
Arbeta med tecken i strängar 12
Jämföra strängar 13
Hämta strängbeteckningar för andra objekt 14
Sammanfoga strängar 14
Returnera delsträngar och mönster i strängar 15
Konvertera strängar mellan versaler och gemener 19
Exempel på strängar: ASCII-bilder 19

Kapitel 3: Arbeta med arrayer
Grunderna i arrayhantering 24
Indexerade arrayer 25
Associativa arrayer 36
Flerdimensionella arrayer 40
Klona arrayer 41
Utökning av klassen Array 42
Exempel på array: PlayList 47

Kapitel 4: Felhantering
Felhanteringens grunder 51
Olika typer av fel 52
Felhantering i ActionScript 3.0 54
Arbeta med felsökningsversioner av Flash-miljöer 56
Hantera synkrona fel i ett program 56
Skapa egna felklasser 61
Svara på felhändelser och status 62
Jämföra felklasserna 65
Exempel på att hantera fel: programmet CustomErrors 69

Kapitel 5: Använda reguljära uttryck
Grunderna för reguljära uttryck 75
Syntaxen för reguljära uttryck 76
Metoder för att använda reguljära uttryck med strängar 89
Exempel på reguljära uttryck: en Wiki-parser 90

Kapitel 6: Arbeta med XML
Grunderna i XML . . . 95
E4X-sättet att arbeta med XML . . . 98
XML-objekt . . . 99
XMLList-objekt . . . 102
Initiera XML-variabler . . . 103
Samla in och omforma XML-objekt . . . 104
Gå igenom XML-strukturer . . . 106
Använda XML-namnutrymmen . . . 110
XML-typkonvertering . . . 111
Läsa externa XML-dokument . . . 113
Exempel på XML i ActionScript: läsa in RSS-data från Internet . . . 113

Kapitel 7: Använda systemspecifik JSON-funktionalitet
Översikt av JSON-API . . . 117
Definiera anpassade JSON-beteenden . . . 118

Kapitel 8: Hantera händelser
Grundläggande om händelsehantering . . . 125
Hur ActionScript 3.0-händelsehantering skiljer sig från tidigare versioner . . . 127
Händelseflödet . . . 129
Händelseobjekt . . . 130
Händelseavlyssnare . . . 134
Exempel på händelsehantering: väckarklocka . . . 141

Kapitel 9: Arbeta med programdomäner

Kapitel 10: Visningsprogrammering
Grunderna i visningsprogrammering . . . 151
Huvudvisningsklasser . . . 154
Fördelar med visningslistan . . . 156
Arbeta med visningsobjekt . . . 158
Ändra visningsobjekt . . . 173
Animera objekt . . . 193
Scenorientering . . . 195
Dynamisk inläsning av visningsinnehåll . . . 198
Exempel på visningsobjekt: SpriteArranger . . . 203

Kapitel 11: Arbeta med geometri
Geometrins grunder . . . 210
Använda Point-objekt . . . 211
Använda Rectangle-objekt . . . 213
Använda Matrix-objekt . . . 216
Exempel på geometri: tillämpa en matrisomformning på ett visningsobjekt . . . 217

Kapitel 12: Använda Drawing API (teckningsgränssnittet)
Grunderna i Drawing API . . . 222
Klassen Graphics . . . 223

Rita linjer och kurvor . . . 223
Rita former med inbyggda metoder . . . 226
Skapa övertoningslinjer och fyllningar . . . 227
Använda klassen Math med ritmetoder . . . 231
Animera med Drawing API . . . 232
Exempel på Drawing API: algoritmisk visuell generator . . . 233
Använda avancerat Drawing API . . . 235

Kapitel 13: Arbeta med bitmappar
Grunderna för arbete med bitmappar . . . 243
Klasserna Bitmap och BitmapData . . . 245
Ändra pixlar . . . 246
Kopiera bitmappsdata . . . 249
Komprimera bitmappdata . . . 250
Skapa texturer med brusfunktioner . . . 251
Rulla bitmappar . . . 253
Utnyttja mipmapping . . . 254
Exempel på bitmapp: animerad snurrande måne . . . 255
Asynkron avkodning av bitmappsbilder . . . 265

Kapitel 14: Filtrera visningsobjekt
Grunderna för filtrering av visningsobjekt . . . 268
Skapa och tillämpa filter . . . 269
Tillgängliga visningsfilter . . . 276
Exempel på filtrering av visningsobjekt: Filter Workbench . . . 293

Kapitel 15: Arbeta med Pixel Bender-skuggningar
Grunderna i Pixel Bender-skuggningar . . . 301
Läsa in eller bädda in en skuggning . . . 303
Få tillgång till skuggningens metadata . . . 304
Ange skuggningsindata och parametervärden . . . 306
Använda en skuggning . . . 311

Kapitel 16: Arbeta med filmklipp
Grunderna för filmklipp . . . 323
Arbeta med MovieClip-objekt . . . 324
Styra uppspelning av filmklipp . . . 324
Skapa MovieClip-objekt med ActionScript . . . 327
Läsa in extern SWF-fil . . . 330
Exempel på filmklipp: RuntimeAssetsExplorer . . . 332

Kapitel 17: Arbeta med rörelseinterpoleringar
Grunderna i rörelseinterpoleringar . . . 336
Kopiera skript för rörelseinterpolering i Flash . . . 337
Infoga skript med rörelseinterpolering . . . 338
Beskriva animeringen . . . 339
Lägga till filter . . . 341
Koppla en rörelseinterpolering till dess visningsobjekt . . . 343

Kapitel 18: Arbeta med omvänd kinematik (IK – inverse kinematics)

Grunderna i omvänd kinematik 344
Översikt över animering av IK-armaturer (Inverse Kinematic) 345
Hämta information om en IK-armatur (Inverse Kinematic) 347
Instansiera en IK Mover och begränsa dess rörelse 347
Flytta en IK-armatur (Inverse Kinematic) 348
Använda fjädring 349
Använda IK-händelser 349

Kapitel 19: Arbeta i tre dimensioner (3D)

Grunderna för 3D-visningsobjekt 351
Om 3D-visningsobjekt i Flash Player och i AIR-miljön 352
Skapa och flytta 3D-visningsobjekt 354
Projicera 3D-objekt i en 2D-vy 356
Exempel: Perspektivprojektion 358
Utföra komplexa 3D-omformningar 360
Använda trianglar för 3D-effekter 363

Kapitel 20: Grunderna för arbete med text

Kapitel 21: Använda klassen TextField

Visa text 373
Markera och ändra text 377
Hämta textinmatningar 378
Begränsa textinmatningar 379
Formatera text 380
Avancerad textåtergivning 384
Arbeta med statisk text 386
Exempel på textfält: Textformatering i tidningsstil 387

Kapitel 22: Använda Flash-textmotorn

Skapa och visa text 397
Hantera händelser i FTE 401
Formatera text 405
Arbeta med teckensnitt 409
Styra text 411
Exempel på Flash Text Engine: nyhetslayout 417

Kapitel 23: Använda Text Layout Framework

Översikt över TLF (Text Layout Framework) 426
Använda Text Layout Framework 427
Strukturera text med TLF 432
Formatera text med TLF 436
Importera och exportera text med TLF 437
Hantera textbehållare med TLF 437
Aktivera markering, redigering och ångra för text med TLF 439
Händelsehantering med TLF 439
Placera bilder i text 439

Kapitel 24: Arbeta med ljud

Grunderna om att arbeta med ljud 441
Ljudarkitekturen 442
Läsa in externa ljudfiler 443
Arbeta med inbäddade ljud 446
Arbeta med direktuppspelade ljudfiler 447
Arbeta med dynamiskt skapat ljud 448
Spela upp ljud 450
Säkerhetsaspekter vid inläsning och uppspelning av ljud 454
Styra ljudvolym och panorering 455
Arbeta med ljudmetadata 456
Använda råa ljuddata 457
Hämta ljudindata 461
Exempel på ljud: Podcast Player 467

Kapitel 25: Arbeta med video

Grunderna för video 474
Videoformat 475
Klassen Video 478
Läsa in videofiler 479
Styra videouppspelning 480
Spela upp video i helskärmsläge 482
Direktuppspelning av videofiler 486
Referenspunkter 486
Skriva callback-metoder för metadata och referenspunkter 487
Använda referenspunkter och metadata 492
Övervaka NetStream-aktivitet 502
Avancerad information för videofiler 505
Exempel på video: videojukebox 507
Använda klassen StageVideo för maskinvaruaccelererad presentation 512

Kapitel 26: Arbeta med kameror

Klassen Camera 521
Visa kamerainnehåll på skärmen 521
Utforma kameraprogram 522
Ansluta till användarens kamera 522
Verifiera att en kamera har installerats 523
Identifiera tillstånd för kameraåtkomst 524
Maximera kamerans videokvalitet 525
Övervaka kamerans status 526

Kapitel 27: Använda DRM (Digital Rights Management)

Förstå arbetsflödet för skyddat innehåll 529
DRM-relaterade medlemmar och händelser i NetStream-klassen 535
Använda klassen DRMStatusEvent 537
Använda klassen DRMAuthenticateEvent 538
Använda klassen DRMErrorEvent 541

Använda klassen DRMManager 542
Använda DRMContentData-klassen 544
Uppdatera Flash Player för stöd av Adobe Access 544
Externa licenser 545
Domänstöd 547
Spela upp krypterat innehåll med domänstöd 547
Förhandsgranska licensen 548
Leverera innehåll 548
Open Source Media Framework 549

Kapitel 28: Lägga till PDF-innehåll i AIR
Identifiera PDF-funktion 551
Läsa in PDF-innehåll 552
Skriptning av PDF-innehåll 552
Kända begränsningar för PDF-innehåll i AIR 554

Kapitel 29: Grunderna i användarinteraktion
Samla in indata från användarna 556
Hantera fokus 557
Identifiera indatatyper 558

Kapitel 30: Tangentbordsindata
Samla in tangentbordsindata 560
Använda klassen IME 562
Virtuella tangentbord 567

Kapitel 31: Musindata
Samla in musindata 575
Exempel på indata från mus: WordSearch 578

Kapitel 32: Indata från pekhändelser, gester och multitouch
Grunder för pekrörelser 583
Identifiering för stöd av pekfunktioner 585
Hantering av pekhändelser 586
Peka och dra 589
Hantering av gesthändelser 590
Felsökning 594

Kapitel 33: Kopiera och klistra in
Grunderna om kopiera och klistra in 597
Läsa från och skriva till systemets Urklipp 598
HTML-kopiering och -inklistring i AIR 598
Format för Clipboard-data 600

Kapitel 34: Indata från accelerometer
Kontrollera stöd för accelerometern 606
Identifiera accelerometerändringar 606

Kapitel 35: Dra och släppa i AIR
Grunderna i dra och släpp i AIR . . . 609
Stöd för utdragningsgesten . . . 611
Stöd för indragningsgesten . . . 614
Dra och släpp i HTML . . . 617
Dra data ut från ett HTML-element . . . 620
Dra data till ett HTML-element . . . 621
Exempel: Åsidosätta standardbeteendet för indragning i HTML . . . 622
Hantera filsläppningar i HTML-sandlådor av annan typ än application . . . 624
Släppa fillöften . . . 625

Kapitel 36: Arbeta med menyer
Grunderna i menyer . . . 634
Skapa inbyggda menyer (AIR) . . . 641
Om snabbmenyer i HTML (AIR) . . . 643
Visa inbygga popup-menyer (AIR) . . . 644
Hantera menyhändelser . . . 644
Exempel på inbyggd meny: Fönster- och programmeny (AIR) . . . 646

Kapitel 37: Aktivitetsfältsikoner i AIR
Om aktivitetsfältsikoner . . . 650
Dockningsikoner . . . 651
Systemfältsikoner . . . 651
Ikoner och knappar i Aktivitetsfältet i Windows . . . 653

Kapitel 38: Arbeta med filsystemet
Använda klassen FileReference . . . 655
Använda filsystems-API:t i AIR . . . 669

Kapitel 39: Lagra lokala data
Delade objekt . . . 703
Krypterad lokal lagringsplats . . . 712

Kapitel 40: Arbeta med lokala SQL-databaser i AIR
Om lokala SQL-databaser . . . 716
Skapa och ändra en databas . . . 720
Ändra data i SQL-databaser . . . 727
Använda synkrona och asynkrona databasåtgärder . . . 755
Använda kryptering med SQL-databaser . . . 760
Strategier för att arbeta med SQL-databaser . . . 777

Kapitel 41: Arbeta med bytearrayer
Läsa och skriva en ByteArray . . . 780
Exempel på ByteArray: Läsa en ZIP-fil . . . 785

Kapitel 42: Grundläggande om nätverk och kommunikation
Nätverksgränssnitt . . . 793
Ändringar i nätverksanslutning . . . 794
DNS-poster (Domain Name System) . . . 796

Kapitel 43: Socketanslutningar
TCP-socketar 799
UDP-socketar (AIR) 810
IPv6-adresser 812

Kapitel 44: HTTP-kommunikation
Läsa in externa data 813
Webbtjänstbegäran 822
Öppna en URL i ett annat program 830

Kapitel 45: Kommunicera med andra Flash Player- och AIR-instanser
Om klassen LocalConnection 832
Skicka meddelanden mellan två program 834
Ansluta till innehåll i andra domäner och till AIR-program 836

Kapitel 46: Kommunicera med interna processer i AIR
Översikt över kommunikation med interna processer 839
Starta och avsluta en intern process 840
Kommunicera med en intern process 841
Säkerhetsaspekter för kommunikation med interna processer 842

Kapitel 47: Använda externt API
Grunderna i externt API 844
Krav och fördelar för det externa API:t 846
Använda klassen ExternalInterface 847
Exempel på extern API: kommunicera mellan ActionScript och JavaScript i en webbläsare 851

Kapitel 48: XML-signaturvalidering i AIR
Grunderna i XML-signaturvalidering 858
Om XML-signaturer 863
Implementera IURIDereferencer-gränssnittet 865

Kapitel 49: Klientsystemmiljö
Grundläggande om klientsystemmiljön 873
Använda klassen System 874
Använda klassen Capabilities 875
Exempel på egenskaper: identifera systemegenskaper 876

Kapitel 50: Anropa och avsluta AIR-program
Anropa program 880
Hämta kommandoradsargument 882
Anropa ett AIR-program när användaren loggar in 885
Anropa ett AIR-program från webbläsaren 886
Avsluta program 888

Kapitel 51: Arbeta med körtids- och operativsystemsinformation i AIR
Hantera filassociationer 890
Hämta körtidsversion och korrigeringsnivå 891

Identifiera AIR-funktioner 891
Spåra användarnärvaro 891

Kapitel 52: Arbeta med inbyggda fönster i AIR
Grunderna om interna fönster i AIR 893
Skapa fönster 900
Hantera fönster 909
Avlyssna fönsterhändelser 918
Visa fönster i helskärmsläge 919

Kapitel 53: Visningsskärmar i AIR
Grunderna om visningsskärmar i AIR 922
Räkna upp skärmarna 923

Kapitel 54: Skriva ut
Grunder för utskrift 926
Skriva ut en sida 927
Utskriftsåtgärder för Flash och för systemet 928
Ställa in storlek, skala och orientering 930
Avancerade utskriftstekniker 932
Exempel på utskrift: flersidesutskrift 935
Exempel på utskrift: skala, beskära och svara 937
Utskriftsexempel: utskriftsformat och utskriftsalternativ 938

Kapitel 55: Geolocation
Identifiera ändringar i geopositionering 941

Kapitel 56: Internationalisering av program
Allmänt om internationalisering av program 945
Översikt över paketet flash.globalization 946
Bestämma språkområde 947
Formatera tal 949
Formatera valutavärden 951
Formatera datum och tid 953
Sortera och jämföra strängar 955
Skiftlägeskonvertering 956
Exempel: Internationalisering av ett börskursprogram 957

Kapitel 57: Lokalisera program
Välja en språkinställning 962
Lokalisera Flex-innehåll 963
Lokalisera Flash-innehåll 963
Lokalisera AIR-program 963
Lokalisera datum, tider och valutor 963

Kapitel 58: Om HTML-miljön
Översikt över HTML-miljön 966
AIR och WebKit 969

Kapitel 59: Programmera HTML och JavaScript i AIR

Om klassen HTMLLoader 985
Undvika säkerhetsrelaterade JavaScript-fel 987
Åtkomst till programmeringsgränssnittsklasser i AIR från JavaScript 992
Om URL:er i AIR 993
Göra ActionScript-objekt tillgängliga för JavaScript 994
Åtkomst till HTML DOM och JavaScript-objekt från ActionScript 996
Bädda in SWF-innehåll i HTML 997
Använda ActionScript-bibliotek i en HTML-sida 1000
Konvertera Date- och RegExp-objekt 1002
Ändra en HTML-formatmall från ActionScript 1002
Korsskriptning av innehåll i olika säkerhetssandlådor 1003

Kapitel 60: Skripta AIR HTML-behållaren

Visningsegenskaper för HTMLLoader-objekt 1008
Rulla HTML-innehåll 1011
Öppna HTML-historiklistan 1012
Ställa in den användaragent som används vid inläsning av HTML-innehåll 1012
Ställa in teckenkodning att använda för HTML-innehåll 1013
Definiera webbläsarliknande användargränssnitt för HTML-innehåll 1013
Skapa underklasser av klassen HTMLLoader 1024

Kapitel 61: Hantera HTML-relaterade händelser i AIR

HTMLLoader-händelser 1025
Hantera DOM-händelser med ActionScript 1026
Svara på JavaScript-undantag som inte fångats upp 1026
Hantera körningshändelser med JavaScript 1028

Kapitel 62: Visa HTML-innehåll i mobilprogram

StageWebView-objekt 1031
Innehåll 1031
Navigeringshändelser 1033
Historik 1034
Fokus 1035
Ta bitmappsbilder 1037

Kapitel 63: Använda arbetare för samtidighet

Arbeta med arbetare och samtidighet 1039
Skapa och hantera arbetare 1040
Kommunicera mellan arbetare 1043

Kapitel 64: Säkerhet

Säkerhet med Flash Platform – översikt 1047
Säkerhetssandlådor 1049
Behörighetskontroller 1053
Begränsa nätverks-API:er 1060
Säkerhet med helskärmsläge 1062
Säkerhet för interaktivt helskärmsläge 1063

Inläsning av innehåll 1064
Korsskriptning 1067
Åtkomst av inlästa medier som data 1070
Inläsning av data 1073
Inläsning av inbäddat innehåll från SWF-filer som har importerats till en säkerhetsdomän 1076
Arbeta med äldre innehåll 1076
Inställning av LocalConnection-behörighet 1077
Kontrollera utgående URL-åtkomst 1077
Delade objekt 1079
Åtkomst av kamera, mikrofon, urklipp, mus och tangentbord 1080
AIR-säkerhet 1081

Kapitel 65: Använda ActionScript-exempel
Exempeltyper 1101
Köra ActionScript 3.0-exempel i Flash Professional 1103
Köra ActionScript 3.0-exempel i Flash Builder 1104
Köra ActionScript 3.0-exempel på mobila enheter 1105

Kapitel 66: SQL-stöd i lokala databaser
SQL-syntax som stöds 1110
Datatyper som stöds 1130

Kapitel 67: SQL-felmeddelanden, ID:n och argument

Kapitel 68: Adobe Graphics Assembly Language (AGAL)
Bytekodsformatet AGAL 1139

Kapitel 1: Arbeta med datum och tidpunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Tidsplanering är oftast en nyckelfaktor i program. I ActionScript 3.0 finns det kraftfulla sätt att hantera kalenderdatum, tidpunkter och tidsintervall. Det finns två huvudklasser som har det mesta för denna tidsplaneringsfunktion: klassen Date och den nya klassen Timer i flash.utils-paketet.

Datum och tidpunkter är en vanlig typ av information som används i ActionScript-program. Du kanske behöver ta reda på den aktuella veckodagen eller hur lång tid en användare tillbringar vid en specifik bildskärm. Med ActionScript kan du använda klassen Date för att representera en enstaka tidpunkt med datum- och tidsuppgifter. Inom en Date-instans finns det värden för specifika datum- och tidsenheter, med bl.a. år, månad, datum, veckodag, timma, minut, sekund, millisekund och tidszon. Dessutom finns den mer avancerade klassen Timer som används för att utföra åtgärder efter en angiven fördröjning eller vid upprepade intervaller.

Fler hjälpavsnitt

Date

flash.utils.Timer

Hantera kalenderdatum och tidpunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla funktioner för hantering av kalenderdatum och tidpunkter i ActionScript 3.0 är samlade i den översta klassen Date. Klassen Date innehåller metoder och egenskaper med vilka du hanterar datum och tidpunkter i antingen universaltid (Coordinated Universal Time) eller lokaltid i en specifik tidszon. Universaltid (UTC) är en standardtidsdefinition som i princip är densamma som GMT (Greenwich Mean Time).

Skapa Date-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Date innehåller en av de mest mångsidiga konstruktormetoderna av alla huvudklasser. Du kan anropa den på fyra olika sätt.

För det första, om inga parametrar anges, returnerar konstruktorn `Date()` ett Date-objekt med aktuellt datum och aktuell tid (lokal tid, baserat på tidszonen). Här är ett exempel:

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

För det andra, om en enda numerisk parameter anges, behandlar konstruktorn `Date()` det som antalet millisekunder sedan 1 januari 1970 och returnerar ett motsvarande Date-objekt. Observera att det millisekundvärde du anger behandlas som antal millisekunder sedan 1 januari 1970 i universaltid (UTC). Däremot visar Date-objektet värden i din lokala tidszon, såvida du inte använder UTC-specifika metoder för att hämta och visa dem. Om du skapar ett nytt Date-objekt med hjälp av en enda millisekundsparameter måste du ta hänsyn till tidzonsskillnaden mellan lokal tid och UTC-tid. I följande kod skapas ett Date-objekt angivet som midnatt den 1 januari 1970, i UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

För det tredje kan du skicka flera numeriska parametrar till konstruktorn `Date()`. Den behandlar parametrarna som år, månad, dag, timma, minut, sekund och millisekund samt returnerar motsvarande Date-objekt. Dessa indataparametrar antas vara i lokal tid, inte i UTC. Med följande kod hämtas ett Date-objekt angivet som midnatt den 1 januari 2000, i lokal tid:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

För det fjärde kan du skicka en enstaka strängparameter till konstruktorn `Date()`. Den försöker tolka strängen som datum- eller tidskomponenter och returnerar sedan ett motsvarande Date-objekt. När du använder det här sättet kan det vara bra att infoga konstruktorn `Date()` i ett `try..catch`-block för att fånga upp tolkningsfel. Konstruktorn `Date()` har stöd för ett antal olika strängformat (som finns listade i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen). Med följande kodsats initieras ett nytt Date-objekt med ett strängvärde:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Om konstruktorn `Date()` inte kan tolka strängparametern genereras inget undantag. Däremot kommer resulterande Date-objekt att innehålla ett ogiltigt datumvärde.

Hämta värden för tidsenheter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan extrahera värden för olika tidsenheter inom ett Date-objekt med hjälp av egenskaper eller metoder i klassen Date. Alla följande egenskaper ger ett värde för en tidsenhet i Date-objektet:

- Egenskapen `fullYear`
- Egenskapen `month`, vilket är i numeriskt format från 0 för januari till 11 för december
- Egenskapen `date`, vilket är kalendernumret på dagen i månaden, från 1 till 31
- Egenskapen `day`, vilket är veckodagen i numeriskt format, där 0 motsvarar söndag
- Egenskapen `hours`, från 0 till 23
- Egenskapen `minutes`
- Egenskapen `seconds`
- Egenskapen `milliseconds`

  Med klassen Date finns det flera sätt att hämta respektive värden på. Du kan till exempel hämta månadsvärdet för ett Date-objekt på fyra olika sätt:

- Egenskapen `month`
- Metoden `getMonth()`
- Egenskapen `monthUTC`
- Metoden `getMonthUTC()`

Alla fyra sätten är i princip likvärdiga vad gäller effektivitet, så du kan använda det sätt som bäst passar för programmet.

Egenskaperna som räknades upp representerar komponenter för det totala datumvärdet. Egenskapen för millisekunder är till exempel aldrig större än 999; när 1 000 uppnås ökas sekundvärdet med 1 och räknaren för millisekunder nollställs.

Om du vill hämta värdet för Date-objektet i antal millisekunder sedan 1 januari 1970 (UTC) så kan du använda metoden `getTime()`. Med den motsvarande metoden `setTime()` kan du ändra värdet för ett befintligt Date-objekt med millisekunder sedan 1 januari 1970 (UTC).

Genomföra beräkningar med datum och tidpunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan utföra addition och subtraktion för datum och tidpunkter med klassen Date. Datumvärden används internt i millisekunder, vilket innebär att du behöver konvertera andra värden till millisekunder innan du kan addera dem till eller subtrahera dem från Date-objekt.

Om programmet ska utföra många beräkningar med datum och tidpunkter kan det vara praktiskt att skapa konstanter för vanliga tidsenheter i antal millisekunder enligt följande:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

På så vis är det enkelt att utföra datumberäkning med standardtidsenheter. I följande kod anges ett datumvärde till en timma från aktuell tidpunkt med hjälp av metoderna `getTime()` och `setTime()`:

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Ett annat sätt att ange ett datumvärde är att skapa ett nytt Date-objekt med en enda millisekundsparameter. Med följande kod läggs 30 dagar till ett datum för att beräkna ett annat:

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

Därefter multipliceras konstanten `millisecondsPerDay` med 30 som motsvarar 30 dagars tid och resultatet läggs till värdet `invoiceDate` och används för att ange värdet `dueDate`.

Konvertering mellan tidszoner

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Beräkningar med datum och tidpunkter är praktiskt om du vill konvertera datum från en tidszon till en annan. Detta görs i metoden `getTimezoneOffset()` som sedan returnerar värdet i antal minuter som Date-objektets tidszon skiljer sig från UTC. Ett värde returneras i minuter eftersom inte alla tidszoner har ökningar i jämna timantal; vissa har en halvtimmes skillnad från närliggande tidszon.

I följande exempel används tidszonsskillnaden för att konvertera ett datum från lokal tid till UTC. Konverteringen utförs först genom att beräkna tidszonsvärdet i millisekunder och sedan justera Date-värdet med antalet:

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

Kontrollera tidsintervall

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du utvecklar program med Adobe Flash CS4 Professional har du tillgång till tidslinjen som ger en jämn talföljd, bildruta för bildruta. I rena ActionScript-projekt måste du däremot förlita dig på andra mekanismer för tidsangivelser.

Programslinga eller timer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I vissa programmeringsspråk måste du göra egna timingscheman med hjälp av slingvillkor som till exempel `for` eller `do..while`.

Programslingor exekveras generellt lika snabbt som den lokala maskinen tillåter, vilket innebär att programmet körs snabbare på vissa maskiner och långsammare på andra. Om programmet kräver ett konsekvent timingintervall behöver du koppla den till en riktig kalender eller klocka. Många program som till exempel spel, animeringar och realtidskontroller kräver vanliga, tidsdrivna mekanismer som är konsekventa från maskin till maskin.

Klassen Timer i ActionScript 3.0 ger en kraftfull lösning. Med händelsemodellen i ActionScript 3.0 skickar klassen Timer händelser varje gång ett angivet tidsintervall uppnås.

Klassen Timer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det bästa sättet att hantera timingfunktioner i ActionScript 3.0 är att använda klassen Timer (flash.utils.Timer), som kan användas för att skicka händelser när ett intervall uppnås.

För att starta en timer skapar du först en instans av klassen Timer, där du anger hur ofta en timerhändelse ska genereras och hur många gånger detta ska ske innan den stoppas.

Med följande kodexempel skapas en Timer-instans som skickar en händelse varje sekund under 60 sekunder:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Objektet Timer skickar objektet TimerEvent varje gång som det givna intervallet uppnås. Händelsetypen för ett TimerEvent-objekt är `timer` (definieras med konstanten `TimerEvent.TIMER`). Objektet TimerEvent innehåller samma egenskaper som standardobjektet Event.

Om Timer-instansen anges med ett fast antal intervaller så skickas även händelsen `timerComplete` (anges med konstanten `TimerEvent.TIMER_COMPLETE`) när det slutliga intervallet uppnås.

Här följer ett litet programexempel som visar hur klassen Timer fungerar:

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

När klassen ShortTimer skapas så skapas en Timer-instans som tickar fram en gång per sekund i fem sekunder. Därefter läggs två avlyssnare till i timern: en som känner av varje tickning och en som känner av händelsen `timerComplete`.

Därefter startas klockningen och från och med då körs metoden `onTick()` med intervaller om en sekund.

Metoden `onTick()` visar helt enkelt aktuellt tickningsantal. Efter fem sekunder körs metoden `onTimerComplete()` som talar om att tiden är slut.

När du kör detta exempel bör du se följande rader på konsolen eller i statusfönstret med en hastighet på en rad i sekunden:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

Timingfunktioner i flash.utils-paketet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript 3.0 innehåller ett antal timingfunktioner liknande de som fanns i ActionScript 2.0. Dessa funktioner finns i flash.utils-paketet och de används precis som i ActionScript 2.0.

Funktion	Beskrivning
`clearInterval(id:uint):void`	Avbryter ett angivet `setInterval()`-anrop.
`clearTimeout(id:uint):void`	Avbryter ett visst `setTimeout()`-anrop.
`getTimer():int`	Returnerar antalet millisekunder som har gått sedan Adobe® Flash® Player eller Adobe® AIR™ startades.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Kör en funktion med ett angivet intervall (i millisekunder).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Kör en angiven funktion efter en angiven fördröjning (i millisekunder).

Dessa klasser finns med i ActionScript 3.0 för bakåtkompatibilitet. Adobe rekommenderar inte att du använder dem i nya program i ActionScript 3.0. Generellt är det enklare och mer effektivt att använda klassen Timer i dina program.

Exempel på datum och tid: enkel analog klocka

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En enkel analog klocka kan få illustrera de här två begreppen för datum och tid:

- Hämtar aktuella datum- och tidpunktsvärden för timme, minut och sekund.
- Använda en timer för att ange tempot för ett program

  Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna för SimpleClock finns i mappen Samples/SimpleClock. Programmet består av följande filer:

Fil	Beskrivning
SimpleClockApp.mxml eller SimpleClockApp.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/simpleclock/SimpleClock.as	Huvudprogramfilen
com/example/programmingas3/simpleclock/AnalogClockFace.as	Ritar en rund urtavla med tim-, minut- och sekundvisare baserat på aktuell tidpunkt.

Definiera klassen SimpleClock

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klockexemplet är enkelt, men det är en god idé att organisera även enkla program så att du enkelt kan utöka dem i framtiden. Hittills används klassen SimpleClock i SimpleClock-programmet för att hantera start- och tidsplaneringsåtgärder. Därefter används en annan klass som heter AnalogClockFace för att visa tiden.

Nedan följer koden som definierar och initierar klassen SimpleClock (observera, att i Flash-versionen utökas i stället klassen Sprite):

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

Klassen har två viktiga egenskaper:

- Egenskapen `face`, som är en instans av klassen AnalogClockFace.
- Egenskapen `ticker`, som är en instans av klassen Timer.

  I klassen SimpleClock används en standardkonstruktor. Metoden `initClock()` tar hand om inställningarna, skapar urtavlan och startar Timer-instansen.

Skapa urtavlan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med följande rader i SimpleClock-koden skapas urtavlan som används för att presentera tiden:

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

Storleken på urtavlan kan skickas till metoden `initClock()`. Om inte värde för `faceSize` skickas så används standardstorleken på 200 pixlar.

Därefter initierar programmet urtavlan och lägger sedan till det till visningslistan via metoden `addChild()`, som ärvs från klassen DisplayObjectContainer. Därefter anropas metoden `AnalogClockFace.draw()` för att visa urtavlan en gång och presentera den aktuella tiden.

Starta timern

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Efter att urtavlan skapats anger metoden `initClock()` en timer:

```
// creates a Timer that fires an event once per second
ticker = new Timer(1000);

// designates the onTick() method to handle Timer events
ticker.addEventListener(TimerEvent.TIMER, onTick);

// starts the clock ticking
ticker.start();
```

Först instansieras en Timer-instans som skickar en händelse en gång per sekund (var 1000:e millisekund). Eftersom ingen andra `repeatCount`-parameter har skickats till konstruktorn `Timer()` upprepas Timer i oändlighet.

Metoden `SimpleClock.onTick()` körs en gång per sekund när händelsen `timer` tas emot:

```
public function onTick(event:TimerEvent):void
{
    // updates the clock display
    face.draw();
}
```

Metoden `AnalogClockFace.draw()` ritar helt enkelt urtavlan och visarna.

Visa aktuell tid

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den mesta koden i klassen AnalogClockFace handlar om inställning av urtavlans visningselement. När AnalogClockFace initieras ritar den en cirkel, placerar en numerisk textetikett vid varje timmarkering och skapar sedan tre Shape-objekt, en för respektive tim-, minut- och sekundvisare.

När SimpleClock-programmet körs anropas metoden `AnalogClockFace.draw()` varje sekund, enligt följande:

```
/**
 * Called by the parent container when the display is being drawn.
 */
public override function draw():void
{
    // stores the current date and time in an instance variable
    currentTime = new Date();
    showTime(currentTime);
}
```

Med den här metoden sparas aktuell tid i en variabel så att tidpunkten inte kan ändras precis när visarna ska flyttas. Därefter anropas metoden `showTime()` för att visa visarna, enligt följande:

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

Först extraheras värdena för timma, minut och sekund motsvarande aktuell tidpunkt. Därefter används dessa värden till att beräkna vinkeln för respektive visare. Eftersom den andra visaren roteras ett helt varv på 60 sekunder så roteras den 6 grader per sekund (360/60). Minutvisaren roteras lika mycket varje minut.

Timvisaren uppdateras varje minut, så den visar en liten förändring allteftersom minuterna tickar. Den roteras 30 grader varje timme (360/12), men den roteras även en halv grad varje minut (30 grader dividerat med 60 minuter).

Kapitel 2: Arbeta med strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen String innehåller metoder som du använder när du arbetar med textsträngar. Strängar är väsentliga i arbetet med många objekt. Metoderna som beskrivs här är praktiska när du arbetar med strängar som används i objekt som t.ex. TextField, StaticText, XML, ContextMenu och FileReference.

En sträng är en sekvens med tecken. I ActionScript 3.0 används ASCII- och Unicode-tecken.

Fler hjälpavsnitt

String

RegExp

parseFloat()

parseInt()

Hur du arbetar med strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I programmeringstermer är en sträng ett textvärde, en sekvens med bokstäver, siffror eller andra tecken intill varandra som utgör ett enda värde. Med den här kodraden skapas en variabel med datatypen String och sedan tilldelas ett stränglitteralvärde till den variabeln:

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

Som visas i exemplet kan du ange ett strängvärde genom att omge det med citat- eller apostroftecken. Här följer fler exempel på strängar:

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Varje gång du ändrar text i ActionScript så arbetar du med ett strängvärde. Klassen String är datatypen du använder i arbetet med textvärden. String-instanser används ofta för egenskaper, metodparametrar o.s.v. i flera andra ActionScript-klasser.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer relaterade till strängar som du kan stöta på:

ASCII Ett system över tecken och symboler för text i datorprogram. I ASCII-systemet används det engelska alfabetet med 26 bokstäver, plus en begränsad uppsättning andra tecken.

Tecken Den minsta enheten textdata (en enskild bokstav eller symbol).

Sammanfogning Sätta ihop flera strängvärden genom att lägga till dem efter varandra och på så vis skapa ett nytt strängvärde.

Tom sträng En sträng som inte varken innehåller text, blanktecken eller andra tecken; skrivs som `""`. Ett tomt strängvärde skiljer sig från en String-variabel med ett null-värde. En null-strängvariabel är en variabel som inte har tilldelats instansen String, medan en tom sträng har en instans med ett värde som inte innehåller några tecken.

String Ett textvärde (teckensekvens).

Stränglitteral Ett strängvärde som används enbart i programkod. Det skrivs som ett textvärde omgivet av citattecken eller apostroftecken.

Delsträng En sträng som är en del av en annan sträng.

Unicode Ett standardsystem över tecken och symboler för text i datorprogram. I Unicode-systemet kan alla tecken från alla skrivsystem användas.

Skapa strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen String används för att representera strängdata (text) i ActionScript 3.0. I ActionScript används både ASCII- och Unicode-tecken. Det enklaste sättet att skapa en sträng är att använda en stränglitteral. När du ska deklarera en stränglitteral använder du apostroftecken (`'`) eller rakt citattecken (`"`). Följande två strängar är likvärdiga:

```
var str1:String = "hello";
var str2:String = 'hello';
```

Du kan även deklarera en sträng med hjälp av operatorn `new`:

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

Följande två strängar ger samma resultat:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Om du ska använda apostroftecken (`'`) inom en stränglitteral som definierats med apostroftecken (`'`) som avgränsare så använder du omvänt snedstreck (`\`) som escape-tecken. Samma sak gäller om du ska använda citattecken (`"`) inom en stränglitteral som definierats med citattecken (`"`) som avgränsare så använder du omvänt snedstreck (`\`) som escape-tecken. Följande två strängar ger samma resultat:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Du kan välja att använda apostroftecken eller citattecken baserat på eventuella apostrof- eller citattecken som redan finns i en stränglitteral:

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Tänk på att ActionScript skiljer mellan ett enstaka apostroftecken (`'`) och ett vänster- eller högerapostroftecken (`'` eller `'`). Samma sak gäller för citattecken. Använd raka citattecken för att markera stränglitteraler. När du klistrar in text från en annan källa i ActionScript måste du se till att använda rätt tecken.

Du kan använda ett omvänt snedstreck (`\`) som escape-tecken för att definiera andra tecken i stränglitteraler, enligt följande:

Escape-sekvens	Tecken
`\b`	Backsteg
`\f`	Arkmatning
`\n`	Radmatning
`\r`	Radbrytning
`\t`	Tabb
`\u`***nnnn***	Unicode-tecknet där teckenkoden har angetts hexadecimalt *nnnn*; t.ex. `\u263a` motsvarar smiley-tecknet.
`\\x`***nn***	ASCII-tecknet med teckenkoden uttryckt hexadecimalt *nn*
`\'`	Apostroftecken
`\"`	Citattecken
`\\`	Omvänt snedstreck

Egenskapen Length

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Varje sträng har en `length`-egenskap vilken är lika med antalet tecken i strängen:

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

En tom sträng och en null-sträng har båda längden noll, enligt följande exempel:

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

Arbeta med tecken i strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Varje tecken i en sträng har en indexposition i strängen (ett heltal). Indexpositionen för det första tecknet är 0. I följande sträng är tecknet `y` i position 0 och tecknet `w` är i position 5:

```
"yellow"
```

Du kan granska enstaka tecken i olika positioner i en sträng med hjälp av metoderna `charAt()` och `charCodeAt()`, enligt följande:

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

När du kör den här koden visas följande utdata:

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

Du kan även använda teckenkoder för att definiera en sträng med metoden `fromCharCode()`:

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

Jämföra strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda följande operatorer när du vill jämföra strängar: `<`, `<=`, `!=`, `==`, `=>` och `>`. Dessa operatorer kan användas med villkorssatser, t.ex. `if` och `while`, enligt följande exempel:

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

När du använder dessa operatorer med strängar granskas teckenkodsvärdet för respektive tecken i strängen och jämför med tecken från vänster till höger:

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Använd operatorerna `==` och `!=` för att jämföra strängar med varandra och med andra objektstyper:

```
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
```

Hämta strängbeteckningar för andra objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan hämta en strängbeteckning för alla typer av objekt. Alla objekt har en `toString()`-metod för denna åtgärd:

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

När du använder operatorn för sammanfogning (+) i en kombination av strängobjekt och icke-strängobjekt så behöver du inte använda metoden `toString()`. Mer information om sammanfogning finns i nästa avsnitt.

Med den globala funktionen `String()` returneras samma värde för ett givet objekt som returneras av objektet som anropar metoden `toString()`.

Sammanfoga strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Sammanfogning av strängar innebär att två strängar sammanfogas sekventiellt till en sträng. Du kan t.ex. använda operatorn + för sammanfogning av två strängar:

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

Du kan även använda operatorn += för att få samma resultat:

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Klassen String innehåller dessutom metoden `concat()` som kan användas så här:

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Om du använder operatorn + (eller operatorn +=) med objektet String och ett objekt som *inte* är en sträng så konverteras automatiskt icke-strängobjektet till ett strängobjekt. På så vis kan uttrycket utvärderas enligt följande:

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

Du kan däremot använda parenteser för att gruppera och tilldela innehåll för operatorn +:

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

Returnera delsträngar och mönster i strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Delsträngar är sekventiella tecken inom en sträng. Exempelvis har strängen `"abc"` följande delsträngar: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. Du kan använda metoder i ActionScript för att söka efter delsträngar i en sträng.

Mönster definieras med strängar eller reguljära uttryck i ActionScript. I följande reguljära uttryck definieras t.ex. ett specifikt mönster; bokstäverna A, B och C följt av ett digitalt tecken (snedstreck är avgränsare i reguljära uttryck):

```
/ABC\d/
```

I ActionScript finns metoder för att söka efter mönster i strängar och ersätta träffarna med delsträngar. Dessa metoder beskrivs i följande avsnitt.

Med hjälp av reguljära uttryck kan du definiera komplicerade mönster. Mer information om publicering finns i ”Använda reguljära uttryck” på sidan 75.

Returnera en delsträng via teckenposition

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoderna `substr()` och `substring()` används på liknande sätt. Båda returnerar en delsträng av en sträng. Båda används med två parametrar. I båda metoderna anger den första parametern startpositionen i strängen. Däremot anger den andra parametern i `substr()`-metoden *längden* på delsträngen som ska returneras och i `substring()`-metoden anger den andra parametern positionen för delsträngens *slut*-tecken (vilket inte ingår i den returnerade strängen). I detta exempel visas skillnaden mellan de båda metoderna:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

Metoden `slice()` fungerar på liknande sätt som metoden `substring()`. Om två positiva heltal anges som parametrar fungerar det på exakt samma sätt. Metoden `slice()` kan däremot användas med negativa heltal som parametrar och då tas teckenpositionen från strängens slut:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

Du kan kombinera icke-negativa och negativa heltal som parametrar i metoden `slice()`.

Returnera teckenpositionen för en matchande delsträng

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda metoderna `indexOf()` och `lastIndexOf()` för att söka efter matchande delsträngar inom en sträng, enligt följande exempel:

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Observera, att metoden `indexOf()` är skiftlägeskänslig.

Du kan ange en andra parameter som anger på vilken indexposition i strängen sökningen ska börja:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

Metoden `lastIndexOf()` söker efter den sista instansen av en delsträng i strängen:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Om du inkluderar en andra parameter med metoden `lastIndexOf()` utförs sökningen från den indexpositionen i strängen och bakåt (höger till vänster):

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

Skapa en array med delsträngar segmenterad via avgränsare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda metoden `split()` för att skapa en array med delsträngar, vilken delas upp baserat på ett avgränsningstecken. Du kan t.ex. segmentera en komma- eller tabbavgränsad sträng i flera strängar.

I följande exempel delas en array upp i delsträngar med et-tecknet (&) som avgränsare:

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

Med den andra, valfria, parametern i metoden `split()` definieras arrayens maximala storlek som returneras.

Du kan även använda ett reguljärt uttryck som avgränsningstecken.

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Läs mer i ”Använda reguljära uttryck” på sidan 75 och Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Returnera mönster i strängar och ersätta delsträngar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I klassen String ingår följande metoder för arbetet med mönster i strängar:

- Använd metoderna `match()` och `search()` för att söka efter delsträngar som matchar ett mönster.
- Använd metoden `replace()` för att söka efter delsträngar som matchar ett mönster och ersätta dem med en angiven delsträng.

Dessa metoder beskrivs i följande avsnitt.

Du kan använda strängar eller reguljära uttryck för att definiera mönster som används i dessa metoder. Mer information om reguljära uttryck finns i ”Använda reguljära uttryck” på sidan 75.

Returnera matchande delsträngar

Metoden `search()` returnerar indexpositionen för den första delsträngen som matchar ett givet mönster, enligt följande exempel:

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

Du kan även använda reguljära uttryck för att definiera mönstret:

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

Utdata för metoden `trace()` är 0, eftersom det första tecknet i strängen är indexpositionen 0. Genom att flaggan `i` anges i det reguljära uttrycket är sökningen inte skiftlägeskänslig.

Metoden `search()` hittar endast en matchning och returnerar motsvarande startindexposition, även om den globala flaggan `g` anges i det reguljära uttrycket.

I följande exempel visas ett mer komplicerat reguljärt uttryck som matchar en sträng inom citattecken:

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

Metoden `match()` fungerar på liknande sätt. Den söker efter en matchande delsträng. När du däremot använder den globala flaggan i ett mönster med reguljärt uttryck så returnerar `match()` en array med matchande delsträngar, enligt följande:

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

Arrayen `results` anges enligt följande:

```
["bob@example.com","omar@example.org"]
```

Mer information om reguljära uttryck finns i ”Använda reguljära uttryck” på sidan 75.

Ersätta matchade delsträngar

Du kan använda metoden `replace()` för att söka efter ett specifikt mönster i en sträng och ersätta matchningar med angiven sträng, enligt följande exempel:

```
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.
```

Observera, att i exemplet är de matchade strängarna inte skiftlägeskänsliga eftersom flaggan `i` (`ignoreCase`) anges i det reguljära uttrycket. Flera matchningar ersätts därför att flaggan `g` (`global`) anges. Mer information om publicering finns i ”Använda reguljära uttryck” på sidan 75.

Du kan inkludera följande `$`-ersättningskoder i ersättningssträngen. Ersättningstexten som visas i följande tabell infogas när ersättningskoden `$`påträffas.

$-kod	Ersättningstext
`$$`	$
`$&`	Den matchande delsträngen.
`$` `	Den del av strängen som föregår den matchande delsträngen. I den här koden används rakt vänsterapostroftecken (`), inte rakt apostroftecken (') eller vänster typografiskt apostroftecken (‘).
`$'`	Den del av strängen som följer den matchande delsträngen. I den här koden används rakt apostroftecken (').
$n	Den *n*:te hämtade parentetiska gruppmatchningen där n är ett ensiffrigt tal (1-9) och $n inte följs av en decimalsiffra.
$nn	Den *nn*:te hämtade parentetiska gruppmatchningen där *nn* är ett tvåsiffrigt decimaltal, 01-99. Om det *nn* :te hämtade är odefinierat är ersättningstexten en tom sträng.

I följande exempel visas användningen av ersättningskoderna `$2` och `$1` som representerar den första och andra hämtade gruppen som matchar varandra:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

Du kan även använda en funktion som den andra parametern i metoden `replace()`. Den matchande texten ersätts med funktionens returnerade värde.

```
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 = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Om du använder en funktion som den andra parametern för metoden `replace()` så skickas följande argument till funktionen:

- Den matchande delen av strängen.
- Eventuella hämtade parentetiska gruppmatchningar. Hur många argument som skickas på det här sättet beror på antalet parentetiska matchningar. Du kan bestämma antalet parentetiska matchningar genom att kontrollera `arguments.length - 3` i funktionskoden.
- Indexpositionen i strängen där matchningen börjar.
- Den fullständiga strängen.

Konvertera strängar mellan versaler och gemener

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Via metoderna `toLowerCase()` och `toUpperCase()` konverteras bokstäverna i en sträng till gemener respektive versaler, enligt följande exempel:

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

Efter att dessa metoder har utförts är källsträngen oförändrad. Om du vill omforma källsträngen skriver du följande:

```
str = str.toUpperCase();
```

Dessa metoder fungerar med utökade tecken, utöver a–z och A–Z:

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

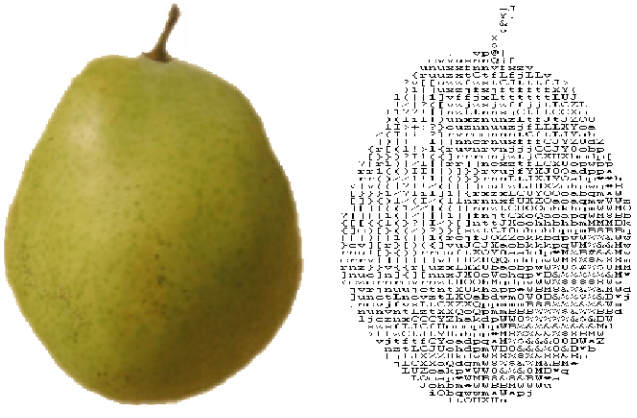
Exempel på strängar: ASCII-bilder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I det här exemplet med ASCII Art visas följande funktioner som används med klassen String i ActionScript 3.0:

- Metoden `split()` i klassen String används för att extrahera värden från en teckenavgränsad sträng (bildinformation i en tabbavgränsad textfil).
- Flera strängändringstekniker där bl.a. `split()`, sammanfogning och extrahering av en del av strängen med hjälp av `substring()` och `substr()` används för att få inledande versal av alla ord i bildrubriker.
- Metoden `getCharAt()` används för att hämta ett enstaka tecken från en sträng (anger ASCII-tecknet som motsvarar ett bitmappsvärde för gråskala).
- Strängsammanfogning används för att skapa en bild bestående av ASCII-tecken.

Termen *ASCII-bildkonst* innebär en textrepresenterad bild som byggs upp av tecken från ett typsnitt med fast teckenbredd, t.ex. Courier New. Följande bild är ett exempel på ASCII-bildkonst som skapats av programmet:



ASCII-versionen av bilden visas till höger.

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna för ASCIIArt finns i mappen Samples/AsciiArt. Programmet består av följande filer:

Fil	Beskrivning
AsciiArtApp.mxml eller AsciiArtApp.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML)
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	Klassen som innehåller programmets huvudfunktioner, inklusive extrahering av bildmetadata från en textfil, inläsning av bilder och hantering av konverteringen för bild-till-text.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	En klass som innehåller metoden `parseBitmapData()` för konvertering av bilddata till en strängversion.
com/example/programmingas3/asciiArt/Image.as	En klass som motsvarar en inläst bitmappsbild.
com/example/programmingas3/asciiArt/ImageInfo.as	En klass som representerar metadata för en ASCII-bild (t.ex. rubrik och bildfilens URL-adress).
bild/	En mapp med bilder som används av programmet.
txt/ImageData.txt	En tabbavgränsad textfil som innehåller information om bilderna som ska läsas in av programmet.

Extrahera tabbavgränsade värden

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I detta exempel används en vanlig rutin där programdata lagras separat från själva programmet. På så vis behövs inte SWF-filen återskapas när data ändras (t.ex. en ytterligare bild läggs till eller en bildrubrik ändras). I det här fallet lagras alla bildmetadata (bl.a. bildrubriken, bildfilens URL-adress och värden som används vid ändring av bilden) i en textfil; txt/ImageData.txt i projektet. Textfilen innehåller följande:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

I filen används ett specifikt tabbavgränsat format. Den första raden är en rubrikrad. Övriga rader innehåller följande data för respektive bitmapp som ska läsas in:

- Filnamnet på bitmappen.
- Visningsnamnet på bitmappen.
- De vita och svarta tröskelvärdena för bitmapparna. Detta är hexvärdena över respektive under vilka en pixel betraktas som helt vit eller helt svart.

När programmet startas görs en inläsning och tolkning av textfilens innehåll via klassen AsciiArtBuilder. På så vis skapas bildstacken som ska visas med hjälp av följande kod från metoden `parseImageInfo()` i klassen AsciiArtBuilder:

```
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)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        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);
    }
}
```

Hela textfilens innehåll finns i en enda String-instans, egenskapen `_imageInfoLoader.data`. Med hjälp av metoden `split()` med radmatningstecknet (`"\n"`) som parameter så delas String-instansen upp i en array (`lines`) vars element utgörs av respektive rad i textfilen. Därefter används en slinga för att bearbeta varje rad (förutom den första som består av enbart rubriker). I den här slingan används metoden `split()` en gång till för att dela upp innehållet i en enstaka rad till en uppsättning med värden (Array-objektet `imageProperties`). I det här fallet används tabbtecknet (`"\t"`) som parameter för metoden `split()`, eftersom värdena i varje rad markeras med tabbtecken.

Normalisera bildrubriker med hjälp av String-metoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En av de utseendemässiga frågorna i detta program är att alla bildrubriker visas i ett standardformat, där alla ord inleds med versal bokstav. Programmet formaterar rubrikraderna allteftersom de extraheras från textfilen, i stället för att anta att textfilen innehåller korrekt formaterade rubriker.

I föregående kodexempel används följande rad vid extrahering av enstaka bildmetadatavärden:

```
        imageInfo.title = normalizeTitle(imageProperties[1]);
```

I den koden skickas textfilens bildrubrik via metoden `normalizeTitle()` innan den sparas i objektet ImageInfo:

```
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(" ");
}
```

I metoden delas rubriken upp till separata ord (avgränsade med blanksteg) med metoden `split()`. Sedan överförs varje ord igenom metoden `capitalizeFirstLetter()` och slutligen sätts orden samman igen till en sträng med metoden `join()` från klassen Array.

Med hjälp av metoden `capitalizeFirstLetter()` ändras varje ord och ges en inledande versal:

```
/**
 * Capitalizes the first letter of a single word, unless it's one of
 * a set of words that are normally not capitalized in English.
 */
private function capitalizeFirstLetter(word:String):String
{
    switch (word)
    {
        case "and":
        case "the":
        case "in":
        case "an":
        case "or":
        case "at":
        case "of":
        case "a":
            // Don't do anything to these words.
            break;
        default:
            // For any other word, capitalize the first character.
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

I det engelska språket kommer *inte* följande ord att få en inledande versal: ”and”, ”the”, ”in”, ”an”, ”or”, ”at”, ”of” eller ”a”. (Detta är en förenklad version av reglerna.) För att tillämpa den här logiken används först en `switch`-sats som kontrollerar om ordet är något av de ord som inte ska ha inledande versal. Om detta stämmer hoppar koden ur `switch`-satsen. Men om ordet ska ges inledande versal görs detta i flera steg:

1. Den första bokstaven i ordet extraheras med `substr(0, 1)`, vilket ger en delsträng som börjar med tecknet på index 0 (anges med den första parametern `0`). Delsträngen blir ett tecken långt (anges med den andra parametern `1`).
2. Detta tecken ändras till en versal med metoden `toUpperCase()`.
3. Återstående tecken i det ursprungliga ordet extraheras med `substring(1)`, som extraherar en delsträng med början på index 1 (den andra bokstaven) till slutet på strängen (anges med att den andra parametern i metoden `substring()` utelämnas).
4. Ordet skapas slutligen genom att den nya versala första bokstaven kombineras med återstående bokstäver med hjälp av strängsammanfogning: `firstLetter + otherLetters`.

Generera ASCII-bildkonst

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen BitmapToAsciiConverter innehåller funktionen som konverterar en bitmappsbild till textrepresentationen i ASCII-tecken. Den här processen utförs med metoden `parseBitmapData()`, som till en del visas här:

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

Med den här koden definieras först stränginstansen `result` som används för att skapa ASCII-versionen av bitmappsbilden. Därefter körs de enskilda pixlarna i källans bitmappsbild i en programslinga. Med hjälp av flera färgändringstekniker konverteras de röda, gröna och blå färgvärdena i en enskild pixel till ett värde i gråskala (en siffra från 0 till 255). Sedan divideras värdet med 4 (se koden) för att konvertera det till ett värde i skalan 0–63, som lagras i variabeln `index`. (Skalan 0–63 används för att ”paletten” med ASCII-tecken som används av programmet innehåller 64 värden.) Den här teckenpaletten definieras som en stränginstans i klassen BitmapToAsciiConverter:

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String =
"@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Eftersom variabeln `index` anger vilket ASCII-tecken i paletten som motsvarar aktuell pixel i bitmappsbilden så hämtas detta tecken från strängen `palette` via metoden `charAt()`. Därefter läggs det till i strängen `result` med hjälp av operatorn för sammanfogning (`+=`). Vid slutet av varje pixelrad sammanfogas dessutom ett radmatningstecken i slutet av strängen `result`, vilket ger en radbrytning för att skapa nästa rad med ”bildtecken”.

Kapitel 3: Arbeta med arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder arrayer för att kunna lagra flera värden i en datastruktur. Du kan använda enkla indexerade arrayer för att lagra värden med hjälp av fasta, ordnade heltalsarrayer eller komplexa associativa arrayer som lagrar värden med hjälp av slumpmässiga nycklar. Arrayer kan också vara flerdimensionella och innehålla element som i sig själva är arrayer. Slutligen kan du använda en vektor för en array, vars element alla är instanser av samma datatyp.

Fler hjälpavsnitt

Array

Vektor

Grunderna i arrayhantering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du programmerar behöver du ofta arbeta med en uppsättning objekt snarare än ett enda objekt. I ett program för uppspelning av musik kanske du måste ha en lista med låtar som väntar på att spelas upp. Då vill du inte behöva skapa en separat variabel för varje låt i listan. Det skulle vara bättre att ha alla Låt-objekt samlade i en bunt, och kunna arbeta med dem som en grupp.

En array är ett programmeringselement som fungerar som en behållare för ett antal objekt, t.ex. en lista med låtar. Det vanligaste är att en array består av instanser av samma klass, men detta är inte något krav i ActionScript. De enskilda objekten i en array kallas *element*. Du kan tänka dig en array som en arkivlåda för variabler. Variabler kan läggas till som element i arrayen, vilket påminner om att lägga mappar i arkivlådan. Du kan arbeta med arrayen som en enskild variabel (som att bära hela arkivlådan till en annan plats). Du kan arbeta med dessa variabler som en grupp (som att bläddra igenom mapparna en och en när du söker efter information). Du kan också få tillgång till dem individuellt (som att öppna arkivlådan och välja en enskild mapp).

Tänk dig att du håller på att skapa en musikspelare där en användare ska kunna välja flera låtar och lägga till dem i spellistan. I ActionScript-koden har du en metod med namnet `addSongsToPlaylist()`, som accepterar en array som en parameter. Oberoende av hur många låtar du lägger till i listan (några, många eller kanske bara en) behöver du bara anropa metoden `addSongsToPlaylist()` en gång och skicka den till arrayen med låtobjekten. I metoden `addSongsToPlaylist()` kan du använda en slinga för att gå igenom alla arrayelementen (låtarna) en efter en och lägga till dem i spellistan.

Den vanligaste typen av ActionScript-array är en *indexerad array*. I en indexerad array lagras varje objekt i ett numrerat fack (kallas *index*). Objekten hämtas med hjälp av ett tal, ungefär som en adress. Indexerade arrayer fungerar bra i de flesta programmeringssituationer. Klassen Array är en vanlig klass som används för att representera en indexerad array.

En indexerad array används ofta för att lagra flera objekt av samma typ (objekt som är instanser av samma klass). Klassen Array har ingen metod för att begränsa vilken typ av objekt den kan innehålla. Klassen Vector är en typ av indexerad array där alla objekt i en enskild array är av samma typ. Om du använder en Vector-instans i stället för en Array-instans blir prestanda bättre och det kan även finnas andra fördelar. Klassen Vector är tillgänglig från och med Flash Player 10 och Adobe AIR 1.5.

Ett särskilt användningsområde för en indexerad array är en *flerdimensionell array*. En flerdimensionell array är en indexerad array, vars element är indexerade arrayer (som i sin tur innehåller andra element).

Ytterligare en arraytyp är *associativa arrayer*, som använder en sträng*nyckel* i stället för ett numeriskt index för att identifiera enskilda element. ActionScript 3.0 innehåller slutligen klassen Dictionary som representerar en *ordlista*. En ordlista är en array som medger att du använder valfri typ av objekt som nyckel för att skilja på olika element.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kommer att stöta på när du programmerar array- och vektorhanteringsrutiner:

Array Ett objekt som används som behållare för att gruppera flera objekt.

Array-åtkomstoperator ([]) Ett par hakparenteser som omger ett index eller en nyckel, som identifierar ett arrayelement. Den här syntaxen används efter ett arrayvariabelnamn för att ange ett enskilt element i arrayen i stället för hela arrayen.

Associativ array En array där strängnycklar används för att identifiera enskilda element.

Bastyp Datatypen för de objekt som kan lagras i en Vector-instans.

Ordlista En array vars objekt består av objektpar som kallas nyckel respektive värde. Nyckeln används i stället för ett numeriskt index för att identifiera ett enskilt element.

Element Ett enskilt objekt i en array.

Index Den numeriska ”adress” som används för att identifiera ett enskilt element i en indexerad array.

Indexerad array Den standardarray som används för att lagra element i en numrerad position och som använder tal (index) för att identifiera enskilda element.

Nyckel Strängen eller objektet som används för att identifiera ett enskilt element i en associativ array eller i en ordlista.

Flerdimensionell array En array som innehåller objekt som är arrayer och inte enskilda värden.

T Standardregeln som används i den här dokumentationen för att representera bastypen för en Vector-instans, oavsett bastyp. T-regeln används för att representera ett klassnamn, vilket visas i beskrivningen av typparametern. (”T” står för ”typ” som i ”datatyp”.)

Type-parameter Den syntax som används med vektorklassnamnet för att ange vektorns bastyp (datatypen för de objekt som lagras i klassen). Syntaxen består av en punkt (.), följt av datatypsnamnet omgivet av vinkelparenteser (`<>`). Sammansatt ser det ut så här: `Vector.<T>`. I den här dokumentationen representeras den klass som anges i type-parametern i allmänhet som `T`.

Vektor En typ av array, vars element alla är instanser av samma datatyp.

Indexerade arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I indexerade arrayer lagras en serie med en eller flera värden organiserade så att varje värde kan hämtas med hjälp av ett heltalsvärde utan tecken. Den första indexposten har alltid talet 0 och indexvärdet ökar med 1 för varje efterföljande element som du lägger till i arrayen. I ActionScript 3.0 används två klasser som indexerade arrayer: klassen Array och klassen Vector.

För indexerade arrayer används ett 32-bitars heltal utan tecken som indexvärde. Maxstorleken för en indexerad array är 2^{32} - 1 eller 4 294 967 295. Ett försök att skapa en array som är större än maxstorleken leder till ett körningsfel.

Om du vill komma åt ett enskilt element i en indexerad array använder du operatorn för arrayåtkomst (`[]`) för att ange indexpositionen för det element som du vill komma åt. Följande kod representerar till exempel det första elementet (elementet vid index 0) i en indexerad array med namnet `songTitles`:

```
songTitles[0]
```

Kombinationen av arrayvariabelnamnet följt av indexet inom hakparentes fungerar som en enskild identifierare. (Med andra ord kan den användas på samma sätt som ett variabelnamn.) Du kan tilldela ett värde till ett indexerat arrayelement genom att använda namnet och indexet på vänster sida i en tilldelningssats:

```
songTitles[1] = "Symphony No. 5 in D minor";
```

Du kan på samma sätt hämta värdet för ett indexerat arrayelement genom att använda namnet och indexet på höger sida i en tilldelningssats:

```
var nextSong:String = songTitles[2];
```

Du kan också använda en variabel inom hakparentes i stället för att ange ett explicit värde. (Variabeln måste innehålla ett heltalsvärde som inte är negativt, ett positivt heltal eller en Number-instans med positivt heltal.) Den här metoden används för det mesta för att ”slinga igenom” elementen i en indexerad array och utföra en åtgärd på några eller alla element. Den här tekniken visas i följande kod. Koden använder en slinga för att få tillgång till varje värde i ett Array-objekt med namnet `oddNumbers`. Den använder programsatsen `trace()` för att skriva ut varje värde i formuläret ”oddNumber[*index*] = *value*”:

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Klassen Array

Den första typen av indexerad array är klassen Array. En Array-instans kan innehålla värden för alla datatyper. Samma Array-objekt kan innehålla objekt som har olika datatyper. En enda Array-instans kan till exempel ha ett String-värde i index 0, en Number-instans i index 1 och ett XML-objekt i index 2.

Klassen Vector

En annan typ av indexerad array som finns i ActionScript 3.0 är klassen Vector. En Vector-instans är en *typbestämd array*, vilket innebär att alla element i en Vector-instans alltid har samma datatyp.

Obs! *Klassen Vector är tillgänglig från och med Flash Player 10 och Adobe AIR 1.5.*

När du deklarerar en Vector-variabel eller instansierar ett Vector-objekt anger du explicit datatypen för objekten som Vector kan innehålla. Den angivna datatypen är dess *bastyp*. Under körning och kompilering (i strikt läge) kontrolleras all kod som ställer in värdet för ett Vector-element eller hämtar ett värde från en Vector. Om datatypen för objektet som läggs till eller hämtas inte motsvarar Vectors bastyp inträffar ett fel.

Förutom datatypsbegränsningen innehåller klassen Vector andra begränsningar som skiljer den från klassen Array:

- En Vector är en kompakt array. Ett Array-objekt kan ha värden i indexen 0 och 7 även om det inte har några värden i position 1 till och med 6. En Vector måste emellertid ha ett värde (eller `null`) i varje index.
- En Vector kan också ha fast längd. Det innebär att antalet element som Vectorn innehåller inte kan ändras.

- Åtkomsten till element tillhörande en Vector är avgränsningskontrollerad. Du kan aldrig läsa ett värde från ett index som är större än det sista elementet (`length` - 1). Du kan aldrig ställa in ett värde med ett index som ligger mer än en position efter det sista indexet. (Med andra ord kan du bara ställa in ett värde vid ett befintligt index eller vid index `[length]`.)

Genom sina begränsningar har en Vector tre stora fördelar gentemot en Array-instans vars element alla är instanser av en enda klass:

- Prestanda: åtkomst och upprepning av arrayelement går mycket snabbare när du använder en Vector-instans än när du använder en Array-instans.
- Typsäkerhet: i strikt läge kan kompileraren identifiera datatypfel. Exempel på sådana fel kan vara att tilldela ett värde med fel datatyp till en Vector eller förvänta fel datatyp när ett värde läses från en Vector. Under körning kontrolleras också datatyperna när data läggs till i eller läses från ett Vector-objekt. Observera emellertid, att när du använder metoderna `push()` eller `unshift()` för att lägga till värden till en Vector kontrolleras inte argumentens datatyper vid kompileringen. När du använder dessa metoder kontrolleras värdena ändå under körning.
- Tillförlitlighet: intervallkontroller av körning (eller kontroller med fast längd) har betydligt högre tillförlitlighet än arrayer.

Förutom dessa begränsningar och fördelar, fungerar klassen Vector ungefär som klassen Array. Ett Vector-objekts egenskaper och metoder liknar, och är i vissa fall identiska med, egenskaper och metoder för en array. I de flesta fall där du skulle kunna använda en array vars element är av samma datatyp, är en Vector-instans att föredra.

Skapa arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda flera olika metoder när du skapar en Array-instans eller Vector-instans. Metoderna för de olika arraytyperna skiljer sig emellertid åt något.

Skapa en Array-instans

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du skapar ett Array-objekt genom att anropa konstruktorn `Array()` eller genom att använda literal Array-syntax.

Konstruktorfunktionen `Array()` kan användas på tre sätt. Du kan till att börja med anropa konstruktorn utan argument och får då en tom array. Du kan använda egenskapen `length` i klassen Array för att bekräfta att arrayen saknar element. I följande exempel anropas konstruktorn `Array()` utan några argument:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

För det andra kan du använda tal som den enda parametern i konstruktorn `Array()`. Nu skapas en array med den angivna längden, där varje elements värde anges som `undefined`. Argumentet måste vara ett heltal utan tecken mellan 0 och 4 294 967 295. I följande exempel anropas konstruktorn `Array()` med ett enda numeriskt argument:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

För det tredje kan du anropa konstruktorn och skicka en lista med element som parametrar. Då skapas en array med de element som motsvarar var och en av parametrarna. I följande exempel skickas tre argument till konstruktorn `Array()`:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

Du kan också skapa arrayer med Array-litteraler. En Array-litteral kan tilldelas direkt till en arrayvariabel som i följande exempel:

```
var names:Array = ["John", "Jane", "David"];
```

Skapa en Vector-instans

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du skapar en Vector-instans genom att anropa konstruktorn `Vector.<T>()`. Du kan också skapa en Vector-instans genom att anropa den globala funktionen `Vector.<T>()`. Den funktionen konverterar ett angivet objekt till en Vector-instans. I Flash Professional CS5 och senare, Flash Builder 4 och senare samt Flex 4 och senare, går det också att skapa en vektorinstans med hjälp av litteral Vector-syntax.

När du deklarerar en Vector-variabel (eller en Vector-metodparameter eller en metodreturtyp) anger du bastypen för Vector-variabeln. Du anger också bastypen när du skapar en Vector-instans genom att anropa konstruktorn `Vector.<T>()`. Med andra ord, när du använder termen `Vector` i ActionScript åtföljs den av en bastyp.

Du anger Vectorns bastyp med typparametersyntax. Typparametern kommer omedelbart efter ordet `Vector` i koden. Den består av en punkt (.), följt av basklassnamnet omgivet av vinkelparenteser (`<>`), vilket visas i detta exempel:

```
var v:Vector.<String>;
v = new Vector.<String>();
```

På exemplets första rad deklareras variabeln `v` som en Vector.`<String>`-instans. Med andra ord representerar den en indexerad array, som bara kan innehålla String-instanser. Den andra raden anropar konstruktorn `Vector()` för att skapa en instans av samma Vector-typ (det vill säga en Vector, vars element alla är String-objekt). Den tilldelar det objektet till `v`.

Använda konstruktorn Vector.<T>()

Om du använder konstruktorn `Vector.<T>()` utan några argument skapas en tom Vector-instans. Du kan testa att en Vector är tom genom att kontrollera dess `length`-egenskap. Följande kod anropar till exempel konstruktorn `Vector.<T>()` utan argument:

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

Om du vet i förväg hur många element som en Vector behöver från början, kan du fördefiniera antalet. Om du vill skapa en Vector med ett visst antal element skickar du antalet element som den första parametern (parametern `length`). Eftersom Vector-element inte kan vara tomma, fylls elementen med instanser av bastypen. Om bastypen är en referenstyp som tillåter `null`-värden innehåller alla element `null`. Annars innehåller alla element standardvärdet för klassen. En uint-variabel kan till exempel inte vara `null`. I följande kodexempel skapas därför Vectorn med namnet `ages` med sju element, där vart och ett innehåller värdet 0:

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

Med konstruktorn `Vector.<T>()` kan du slutligen också skapa en Vector med fast längd genom att skicka `true` för den andra parametern (parametern `fixed`). I så fall skapas Vectorn med angivet antal element, vilket inte kan ändras. Du kan emellertid fortfarande ändra värdena på elementen i en Vector med fast längd.

Använda konstruktorn för litteral Vector-syntax

I Flash Professional CS5 och senare, Flash Builder 4 och senare samt Flex 4 och senare kan du ange en lista med värden till konstruktorn för `Vector.<T>()` för att ange Vectorns startvärden:

```
// var v:Vector.<T> = new <T>[E0, ..., En-1 ,];
// For example:
var v:Vector.<int> = new <int>[0,1,2,];
```

Följande information gäller för denna syntax:

- Det efterföljande kommat är valfritt.
- Tomma objekt i arrayen stöds inte. En sats som `var v:Vector.<int> = new <int>[0,,2,]` genererar till exempel ett kompileringsfel.
- Det går inte att ange en standardlängd för Vector-instansen. I stället är längden densamma som antalet element i initieringslistan.
- Det går inte att ange om Vector-instansen har en fast längd. Använd i stället egenskapen `fixed`.
- Dataförlust eller fel kan inträffa om objekt som anges som värden inte stämmer överens med den angivna typen. Till exempel:

  ```
  var v:Vector.<int> = new <int>[4.2]; // compiler error when running in strict mode
  trace(v[0]); //returns 4 when not running in strict mode
  ```

Använda konstruktorn Vector.<T>().

Förutom konstruktorn `Vector.<T>()` och konstruktorn för litteral Vector-syntax kan du använda den globala funktionen `Vector.<T>()` när du skapar ett Vector-objekt. Den globala funktionen `Vector.<T>()` är en konverteringsfunktion. När du anropar den globala funktionen `Vector.<T>()` anger du bastypen för den Vector som metoden returnerar. Du skickar en enda indexerad array (Array- eller Vector-instans) som ett argument. Metoden returnerar sedan en Vector med angiven bastyp som innehåller värdena i källarrayargumentet. Följande kodexempel visar syntaxen för anrop av den globala funktionen `Vector.<T>()`:

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

Den globala funktionen `Vector.<T>()` utför datatypskonvertering på två nivåer. För det första, när en Array-instans skickas till funktionen returneras en Vector-instans. För det andra, när källarrayen är en Array- eller Vector-instans försöker funktionen att konvertera källarrayens element till värden av bastypen. Vid konverteringen används de vanliga konverteringsreglerna för ActionScript-datatyper. Följande kodexempel konverterar String-värdena i källarrayen till heltal i den resulterande Vectorn. Decimaldelen av det första värdet (`"1.5"`) kortas av, och det ickenumeriska tredje värdet (`"Waffles"`) konverteras till 0 i resultatet:

```
var numbers:Vector.<int> = Vector.<int>(["1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

Om något av källelementen inte kan konverteras inträffar ett fel.

När koden anropar den globala funktionen `Vector.<T>()`, och ett element i källarrayen är en instans av en underklass i den angivna bastypen, läggs elementet till i den resulterande vektorn (inget fel inträffar). Att använda den globala funktionen `Vector.<T>()` är det enda sättet att konvertera en vektor med bastypen `T` till en vektor med en bastyp som är en överordnad klass till `T`.

Infoga arrayelement

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det vanligaste sättet att lägga till ett element i en indexerad array är att använda operatorn för arrayåtkomst (`[]`). Om du vill ställa in värdet för ett indexerat arrayelement ska du använda Array- eller Vector-objektnamnet och indexnumret på vänster sida i en tilldelningssats:

```
songTitles[5] = "Happy Birthday";
```

Om en Array eller Vector inte redan har ett element vid detta index, skapas indexet och värdet lagras där. Om det finns ett värde vid detta index, ersätter det nya värdet det befintliga.

Med ett Array-objekt kan du skapa ett element vid valfritt index. Med ett Vector-objekt kan du emellertid bara tilldela ett värde till ett befintligt index eller till nästa tillgängliga index. Nästa tillgängliga index motsvarar Vector-objektets `length`-egenskap. Det säkraste sättet att lägga till ett nytt element i ett Vector-objekt är att använda kod som den nedan:

```
myVector[myVector.length] = valueToAdd;
```

Tre av metoderna i klasserna Array och Vector – `push()`, `unshift()` och `splice()` – används för att infoga element i en indexerad array. Metoden `push()` lägger till ett eller flera element i slutet av en array. Detta innebär att det element som senast infogades i arrayen med metoden `push()` får det högsta indexvärdet. Med metoden `unshift()` infogas ett eller flera element i början av en array, vilket alltid är vid indexvärdet 0. Metoden `splice()` används för att infoga valfritt antal objekt vid ett angivet index i arrayen.

I följande exempel visas alla tre metoderna. En array med namnet `planets` skapas för att lagra namnen på planeterna i den ordning de ligger från solen. Till att börja med anropas metoden `push()` för att lägga till det första objektet, `Mars`. Därefter anropas metoden `unshift()` för att lägga till ett objekt, `Mercury`, som ska ligga först i arrayen. Slutligen anropas metoden `splice()` för att infoga objekten `Venus` och `Earth` efter `Mercury`, men före `Mars`. Det första argumentet som skickas till `splice()`, heltalet 1, anger att infogningen ska börja vid index 1. Det andra argumentet som skickas till `splice()`, heltalet 0, anger att inga objekt ska raderas. Det tredje och fjärde argumentet som skickas till `splice()`, `Venus` och `Earth`, är de objekt som ska infogas.

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

Metoderna `push()` och `unshift()` returnerar båda ett heltal utan tecken, som representerar längden på den ändrade arrayen. Metoden `splice()` returnerar en tom array när den används för att infoga element, vilket kan tyckas underligt, men blir mer begripligt i ljuset av den flexibilitet som metoden `splice()` uppvisar. Du kan använda metoden `splice()` både för att infoga element i en array och för att ta bort element från en array. När du använder metoden `splice()` för att ta bort element returneras en array som innehåller de borttagna elementen.

Obs! *Om ett Vector-objekts `fixed`-egenskap är `true` kan det totala antalet element i vektorn inte ändras. Om du försöker att lägga till ett nytt element i en Vector med fast längd med de metoder som beskrivs här, inträffar ett fel.*

Hämta värden och ta bort arrayelement

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det enklaste sättet att hämta värdet på ett element från en indexerad array är att använda operatorn för arrayåtkomst (`[]`). Om du vill hämta värdet för ett indexerat arrayelement ska du använda Array- eller Vector-objektnamnet och indexnumret på höger sida i en tilldelningssats:

```
var myFavoriteSong:String = songTitles[3];
```

Det går att försöka hämta ett värde från en Array eller Vector genom att använda ett index där det inte finns några element. I så fall returnerar ett Array-objekt värdet odefinierat och en Vector returnerar ett RangeError-undantag.

Tre metoder i klasserna Array och Vector – `pop()`, `shift()` och `splice()` – används för att ta bort element. Metoden `pop()` används för att ta bort ett element från slutet av arrayen. Den tar med andra ord bort elementet med det högsta indexvärdet. Metoden `shift()` används för att ta bort ett element från början av arrayen, vilket innebär att den alltid tar bort elementet vid indexvärdet 0. Metoden `splice()`, som också kan användas för att infoga element, tar bort ett godtyckligt antal element med början från det indexvärde som anges i det första argumentet som skickas till metoden.

I följande exempel används alla tre metoder för att ta bort element från en Array-instans. En array med namnet `oceans` skapas för att lagra namnen på de stora världshaven. Vissa av namnen i arrayen är sjöar och inte världshav, så dessa måste tas bort.

Först används metoden `splice()` för att ta bort objekten `Aral` och `Superior` och för att infoga `Atlantic` och `Indian`. Det första argumentet som skickas till `splice()`, heltalet 2, anger att åtgärden ska starta med det tredje objektet i listan, alltså vid index 2. Det andra argumentet, 2, anger att två objekt ska tas bort. De återstående argumenten, `Atlantic` och `Indian`, är värden som ska infogas vid index 2.

Därefter används metoden `pop()` för att ta bort det sista elementet i arrayen, `Huron`. Slutligen används metoden `shift()` för att ta bort det första elementet i arrayen, `Victoria`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

Metoderna `pop()` och `shift()` returnerar båda det objekt som togs bort. Datatypen för Array-instansens returnerade värde är Object eftersom arrayer kan innehålla värden av alla datatyper. För en Vector-instans är datatypen för det returnerade värdet bastypen för Vectorn. Metoden `splice()` returnerar en array eller vektor som innehåller de borttagna värdena. Du kan ändra arrayexemplet `oceans` så att anropet till `splice()` tilldelar den returnerade arrayen till en ny arrayvariabel, enligt följande:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

Du kan träffa på kod där operatorn `delete` används på ett element i ett Array-objekt. Operatorn `delete` anger värdet för ett arrayelement som `undefined`, men elementet tas inte bort från arrayen. I följande exempel används operatorn `delete` på det tredje elementet i arrayen `oceans`, men längden på arrayen är fortfarande 5:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

Du kan korta av en array eller vektor med hjälp av en arrays `length`-egenskap. Om du ställer in egenskapen `length` för en indexerad array på en längd som är mindre än den aktuella längden på arrayen, kortas arrayen av och de element som lagras vid indexnummer högre än det nya värdet på `length` minus 1 tas bort. Om till exempel arrayen `oceans` sorteras så att alla giltiga poster finns i början av arrayen, kan du använda egenskapen `length` för att ta bort posterna i slutet av arrayen, vilket visas i följande kod:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

Obs! *Om ett Vector-objekts `fixed`-egenskap är `true` kan det totala antalet element i vektorn inte ändras. Om du försöker att ta bort ett element från eller korta av en Vector med fast längd med de metoder som beskrivs här, inträffar ett fel.*

Sortera en array

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns tre metoder – `reverse()`, `sort()` och `sortOn()` – som används för att ändra ordningen i en indexerad array genom att sortera den på olika sätt. Alla dessa metoder används för att förändra en befintlig array. I tabellen nedan sammanfattas dessa metoder och deras beteende för Array- och Vector-objekt:

Metod	Beteende hos Array	Beteende hos Vector
`reverse()`	Ändrar ordningen på elementen så att det sista elementet blir det första, det näst sista det andra och så vidare	Identiskt med beteendet för Array
`sort()`	Med denna kan du sortera arrayens element på olika fördefinierade sätt, till exempel i bokstavs- eller nummerordning. Du kan också ange en anpassad sorteringsalgoritm.	Sorterar elementen enligt den anpassade sorteringsalgoritmen som du anger
`sortOn()`	Med denna kan du sortera objekt som har en eller flera gemensamma egenskaper, och ange egenskapen eller egenskaperna som du vill använda för att sortera nycklar	Finns inte för klassen Vector

Metoden reverse()

Metoden `reverse()` har inga parametrar och returnerar inget värde. Du använder den för att kasta om ordningen i arrayen från den aktuella till den omvända. I följande exempel kastas ordningen om på de världshav som finns i arrayen `oceans`:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

Grundläggande sortering med metoden sort() (enbart klassen Array)

För en Array-instans ordnar metoden `sort()` elementen i en array med hjälp av *standardsorteringsordningen*. En standardsorteringsordning kännetecknas av följande:

- Sorteringen är skiftlägeskänslig, vilket betyder att versaler sorteras före gemener. Till exempel kommer bokstaven D före bokstaven b.
- Sorteringen är stigande vilket innebär att lägre teckenkoder (som A) kommer före högre teckenkoder (som B).
- Sorteringen placerar identiska värden intill varandra utan någon speciell ordning.
- Sorteringen är strängbaserad, vilket innebär att element konverteras till strängar innan de jämförs (till exempel kommer 10 före 3 eftersom strängen `"1"` har lägre teckenkod än strängen `"3"`).

Det kan emellertid finnas tillfällen då du vill sortera arrayen utan hänsyn till versaler eller gemener, i fallande ordning eller så kanske du vill att arrayer som innehåller siffror ska sorteras numeriskt och inte alfabetiskt. Klassen Arrays `sort()`-metod har en `options`-parameter, som används för att ändra förutsättningarna för standardsorteringsordningen. Alternativen definieras med hjälp av ett antal statiska konstanter i klassen Array enligt följande lista:

- `Array.CASEINSENSITIVE`: Detta alternativ använder du när du inte vill ta hänsyn till versaler och gemener. Nu kommer gemenen b att komma före versalen D.

- `Array.DESCENDING`: Detta alternativ leder till fallande sorteringsordning. Stigande är standard. Bokstaven B kommer till exempel före bokstaven A.
- `Array.UNIQUESORT`: Detta alternativ leder till att sorteringen avbryts om två identiska värden hittas.
- `Array.NUMERIC`: Detta alternativ används för numerisk sortering så att 3 kommer före 10.

I följande exempel visas några av dessa alternativ. En array med namnet `poets` skapas och sorteras med flera olika alternativ.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

Anpassad sortering med metoden sort() (klassen Array och klassen Vector)

Förutom den grundläggande sorteringen som finns för ett Array-objekt kan du också definiera en anpassad sorteringsregel. Den här tekniken är den enda formen av `sort()`-metoden som finns för klassen Vector. Du definierar en anpassad sortering genom att skriva en anpassad sorteringsfunktion och skicka den som ett argument till metoden `sort()`.

Om du exempelvis har en lista med namn där varje listelement består av personens fullständiga namn, men där du vill sortera listan efter efternamnet, måste du använda en anpassad sorteringsfunktion för att tolka varje element och använda efternamnet i sorteringsfunktionen. I följande exempel visas hur detta kan göras med en anpassad funktion som används som en parameter i metoden `Array.sort()`:

```
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); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

I den anpassade sorteringsfunktionen `orderLastName()` används ett reguljärt uttryck för att extrahera efternamnet från varje element för jämförelseoperationen. Funktionsidentifieraren `orderLastName` används som den enda parametern när metoden `sort()` anropas i arrayen `names`. Sorteringsfunktionen accepterar två parametrar, `a` och `b`, eftersom den arbetar med två arrayelement samtidigt. Sorteringsfunktionens returnerade värde anger hur elementen ska sorteras:

- Om returvärdet är -1 visar det att den första parametern, `a`, kommer före den andra parametern, `b`.
- Om returvärdet är 1 visar det att den andra parametern, `b`, kommer före den första, `b`.
- Om returvärdet är 0 visar det att elementen har samma sorteringsprioritet.

Metoden sortOn() (enbart klassen Array)

Metoden `sortOn()` är utformad för Array-objekt med element som innehåller objekt. Dessa objekt förväntas ha minst en gemensam egenskap, som kan användas som sorteringsnyckel. Om du använder metoden `sortOn()` för andra typer av arrayer kommer du att få oväntade resultat.

Obs! *Klassen Vector har ingen `sortOn()`-metod. Den här metoden finns bara för Array-objekt.*

I följande exempel revideras arrayen `poets` så att varje element blir ett objekt i stället för en sträng. Varje objekt kommer nu att innehålla både poetens efternamn och födelseår.

```
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"});
```

Du kan använda metoden `sortOn()` för att sortera arrayen efter egenskapen `born`. Metoden `sortOn()` definierar två parametrar, `fieldName` och `options`. Argumentet `fieldName` måste specificeras som en sträng. I följande exempel anropas `sortOn()` med två argument, ”`born"` och `Array.NUMERIC`. Argumentet `Array.NUMERIC` används för att se till att sorteringen görs numeriskt i stället för alfabetiskt. Detta är en bra vana även när alla tal har samma antal siffror, eftersom det ser till att sorteringen kommer att fungera som du förväntar dig även om ett tal med färre eller fler siffror läggs till arrayen senare.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Sortera utan att ändra den ursprungliga arrayen (enbart klassen Array)

Metoderna `sort()` och `sortOn()` används vanligtvis för att förändra en array. Om du vill sortera en array utan att ändra den befintliga arrayen skickar du konstanten `Array.RETURNINDEXEDARRAY` som en del av parametern `options`. Detta leder till att metoden returnerar en ny array som visar sorteringen och som lämnar den ursprungliga arrayen oförändrad. Arrayer som returneras av metoder är enkla arrayer med indexvärden som visar den nya sorteringsordningen och som inte innehåller några element från den ursprungliga arrayen. Om du till exempel vill sortera arrayen `poets` efter födelseår utan att ändra arrayen, tar du med konstanten `Array.RETURNINDEXEDARRAY` som en del av det argument som skickas för parametern `options`.

I följande exempel lagras den returnerade indexinformationen i en array med namnet `indices`. Därefter används arrayen `indices` i kombination med den oförändrade arrayen `poets` för att ordna poeterna efter födelseår:

```
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);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Ställa frågor mot en array

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Fyra metoder i klasserna Array och Vector – `concat()`, `join()`, `slice()` och `toString()` – används för att ställa frågor mot arrayen och inte för att förändra den. Metoderna `concat()` och `slice()` returnerar båda nya arrayer, medan metoderna `join()` och `toString()` båda returnerar strängar. I metoden `concat()` tas en ny array eller lista med element som argument och kombinerar den med en befintlig array för att skapa en ny array. Metoden `slice()` har två parametrar med de passande namnen `startIndex` och `endIndex`. Den returnerar en ny array som innehåller en kopia av de element som delades upp från den befintliga arrayen. Uppdelningen börjar med elementet vid `startIndex` och slutar med elementet precis före `endIndex`. Detta kanske behöver upprepas: elementet vid `endIndex` ingår INTE i returvärdet.

I följande exempel används `concat()` och `slice()` för att skapa nya arrayer med hjälp av element från andra arrayer:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

Du kan använda metoderna `join()` och `toString()` för att ställa frågor mot arrayen och returnera dess innehåll som en sträng. De båda metoderna är identiska om inga parametrar används för metoden `join()`, eftersom de returnerar en sträng med en kommaseparerad lista med alla element i strängen. För metoden `join()`, till skillnad från metoden `toString()`, accepteras en parameter med namnet `delimiter`, vilket innebär att du kan välja vilken symbol som ska användas som avgränsare mellan elementen i den returnerade strängen.

I nästa exempel skapas en array med namnet `rivers` som anropar både `join()` och `toString()` för att returnera värdena i arrayen som en sträng. Metoden `toString()` används för att returnera kommaseparerade värden (`riverCSV`), medan metoden `join()` används för att returnera värden som separeras med tecknet `+`.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

En sak som du bör tänka på när du använder metoden `join()` är alla kapslade Array- och Vector-instanser alltid returneras med kommaavgränsade värden, oavsett vilken avgränsare du anger för elementen i huvudarrayen, vilket framgår av följande exempel:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

Associativa arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I en associativ array, som ibland även kallas *hash* eller *map*, används *nycklar* i stället för numeriska index för att ordna lagrade värden. Varje nyckel i en associativ array är en unik sträng som används för att få åtkomst till lagrade värden. En associativ array är en instans av klassen Object, vilket innebär att varje nyckel motsvarar ett egenskapsnamn. Associativa arrayer är osorterade samlingar av par med nycklar och värden. I koden förutsätts inte att nycklarna för en associativ array ligger i någon särskild ordning.

I ActionScript 3.0 ingår också en avancerad typ av associativ array, som kallas för *ordlista*. Ordlistor, som är instanser av klassen Dictionary i flash.utils-paketet, använder nycklar som kan vara av valfri datatyp. Med andra ord är Dictionary-nycklarna inte begränsade till värden av typen String.

Associativa arrayer med strängnycklar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns två sätt att skapa associativa arrayer i ActionScript 3.0: Det första är att använda en Object-instans. Genom att använda en Object-instans kan du initiera arrayen med en objektlitteral. En instans av Object-klassen, som även kallas ett *generiskt objekt*, är funktionsmässigt identisk med en associativ array. Varje egenskapsnamn för det generiska objektet fungerar som en nyckel som ger åtkomst till ett sparat värde.

I exemplet nedan skapas en associativ array med namnet `monitorInfo` och här används en objektlitteral för att initiera arrayen med två nyckel-/värde-par:

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Om du inte behöver initiera arrayen när den deklareras kan du använda konstruktorn Object för att skapa arrayen, vilket visas i nästa exempel:

```
var monitorInfo:Object = new Object();
```

När arrayen har skapats, antingen med hjälp av en objektlitteral eller konstruktorn för klassen Object, kan du lägga till nya värden i arrayen med hjälp av operatorn för arrayåtkomst (`[]`) eller punktoperatorn (`.`). I följande exempel läggs två nya värden till i `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

Observera att nyckeln med namnet `aspect ratio` innehåller ett blanksteg. Detta fungerar med operatorn för arrayåtkomst (`[]`), men leder till ett fel med punktoperatorn. Därför bör du inte använda blanksteg i nyckelnamn.

Det andra sättet att skapa en associativ array är att använda konstruktorn Array (eller konstruktorn för någon dynamisk klass) och sedan antingen operatorn för arrayåtkomst (`[]`) eller punktoperatorn (`.`) för att lägga till par med nyckel och värde till arrayen. Om du deklarerar att den associativa arrayen är av typen Array, kan du inte använda en objektlitteral för att initialisera den. I följande exempel skapas en associativ array med namnet `monitorInfo` med hjälp av konstruktorn Array, och nycklar med namnen `type` och `resolution` läggs till tillsammans med respektive värden:

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Det finns ingen fördel med att använda kontruktorn Array för att skapa en associativ array. Du kan inte använda egenskapen `Array.length` eller någon annan metod för klassen Array med associativa arrayer, även om du använder konstruktorn Array eller datatypen Array. Konstruktorn Array fungerar bäst när du använder den för att skapa indexerade arrayer.

Associativa arrayer med objektnycklar (ordlistor)

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen Dictionary kan du skapa en associativ array där objekt används som nycklar och inte strängar. Sådana arrayer kallas ibland ordlistor, hash-tabeller eller ”maps”. Tänk dig ett program som används för att bestämma platsen för ett Sprite-objekt utifrån dess association till en speciell behållare. Du kan då använda ett Dictionary-objekt för att mappa varje Sprite-object till en behållare.

I exemplet nedan skapas tre instanser av klassen Sprite för att utgöra nycklar för objektet Dictionary. Varje nyckel tilldelas ett värde på antingen `GroupA` eller `GroupB`. Värdena kan ha alla datatyper, men i detta exempel är både `GroupA` och `GroupB` instanser av klassen Object. Därav följer att du får åtkomst till värdet som associeras med respektive nyckel via operatorn för arrayåtkomst (`[]`), vilket framgår av följande kod:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
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");
}
```

Iterera med objektnycklar

Du kan iterera igenom innehållet i ett Dictionary-objekt med antingen en `for..in`-slinga eller en `for each..in`-slinga. Med en `for..in`-slinga itererar du med hjälp av nycklar, medan du för en `for each..in`-slinga använder värdet som associerats med varje nyckel.

Du använder `for..in`-slingan för att få direkt åtkomst till objektnycklarna i ett Dictionary-objekt. Du kan även få åtkomst till värden i objektet Dictionary med operatorn för arrayåtkomst (`[]`). I följande kod används det tidigare exemplet med `groupMap`-ordlistan för att visa hur du itererar igenom ett Dictionary-objekt med `for..in`-slinga:

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Du använder `for each..in`-slingan för att få direkt åtkomst till värden i ett Dictionary-objekt. I följande kod används exemplet med `groupMap`-ordlistan för att visa hur du itererar igenom ett Dictionary-objekt med `for each..in`-slinga:

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

Objektnycklar och minneshantering

I Adobe® Flash® Player och Adobe® AIR™ används ett system för att samla ihop skräp för att rensa minnet från sådant som inte längre behövs. När det inte längre finns några referenser som pekar mot ett objekt uppfyller objektet villkoren för att betraktas som skräp och det kommer då att rensas bort nästa gång som skräpsamlingen utförs. I nästa exempel skapas ett nytt objekt och variabeln `myObject` tilldelas en referens till objektet:

```
var myObject:Object = new Object();
```

Så länge det finns några referenser till objektet går det inte att använda skräpsamlingssystemet för att återställa den minneskapacitet som objektet upptar. Om värdet för `myObject` ändras så att det pekar på ett annat objekt eller anges som `null` blir det minnesutrymme, som upptogs av det ursprungliga objektet, tillgängligt för skräpsamling. Detta förutsätter dock att det inte finns några andra referenser till det ursprungliga objektet.

Om du använder `myObject` som en nyckel i ett Dictionary-objekt skapar du en ny referens till det ursprungliga objektet. I följande exempel skapas två referenser till ett objekt – variabeln `myObject` och nyckeln i objektet `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Om du vill att det objekt som `myObject` refererar till ska vara tillgängligt för skräpsamling måste du ta bort alla referenser till det. I detta fall måste du ändra värdet för `myObject` och radera nyckeln `myObject` från `myMap`, vilket framgår av följande exempel:

```
myObject = null;
delete myMap[myObject];
```

Du kan också använda parametern `useWeakReference` i Dictionary-konstruktorn för att göra alla Dictionary-nycklar till *svaga referenser*. Skräpsamlingssystemet ignorerar svaga referenser, vilket innebär att ett objekt som endast innehåller svaga referenser kan tas bort vid en skräpsamling. I nästa exempel behöver du inte ta bort nyckeln `myObject` från `myMap` för att göra objektet tillgängligt för skräpsamling:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

Flerdimensionella arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flerdimensionella arrayer har andra arrayer som element. Tänk dig en lista med uppgifter som är sparad som en indexerad array med strängar:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Om du vill spara en separat lista med uppgifter för varje dag i veckan ska du skapa en flerdimensionell array med ett element för varje veckodag. Varje element innehåller en indexerad array, som liknar arrayen `tasks`, som innehåller listan med uppgifter. Du kan använda alla kombinationer av indexerade och associativa arrayer i flerdimensionella arrayer. I exemplen i följande avsnitt används antingen två indexerade arrayer eller en associativ array av indexerade arrayer. Du kan själv prova de andra kombinationerna.

Två indexerade arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du använder två indexerade arrayer kan resultatet visas som en tabell eller ett kalkylark. Elementen i den första arrayen representerar raderna i tabellen, medan elementen för den andra arrayen representerar kolumnerna.

I följande flerdimensionella array används två indexerade arrayer för att spåra uppgiftslistor för varje dag i veckan. Den första arrayen, `masterTaskList`, skapas med konstruktorn för klassen Array. Varje element i arrayen representerar en dag i veckan, där index 0 representerar måndag och index 6 söndag. Dessa element kan du se som rader i tabellen. Du skapar uppgiftslistan för varje dag genom att tilldela en arraylitteral till vart och ett av de sju elementen som du skapar i arrayen `masterTaskList`. Arraylitteralerna representerar kolumnerna i tabellen.

```
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"];
```

Du kan använda operatorn för arrayåtkomst (`[]`) för att komma åt de enskilda objekten i uppgiftslistorna. Den första uppsättningen med hakparenteser representerar veckodagen och den andra uppsättningen representerar uppgiftslistan för den aktuella dagen. Om du till exempel vill hämta den andra uppgiften från onsdagslistan börjar du med att använda index 2 för onsdag och därefter index 1 för den andra uppgiften i listan.

```
trace(masterTaskList[2][1]); // output: dentist
```

Om du vill hämta den första uppgiften från söndagslistan använder du index 6 för söndag och index 0 för den första uppgiften i listan.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

Associativa arrayer med en indexerad array

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill göra enskilda arrayer enklare att komma åt ska du använda en associativ array för veckodagarna och en indexerad array för uppgiftslistan. När du använder associativa arrayer kan du använda punktsyntaxen för att referera till en speciell veckodag, men till kostnad av extra bearbetningskapacitet för att komma åt elementen i den associativa arrayen. I följande exempel används en associativ array som bas för en uppgiftslista, med ett nyckel- och värdepar för varje veckodag:

```
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"];
```

Punktsyntaxen gör koden mer läsvänlig eftersom du inte behöver använda flera uppsättningar hakparenteser.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

Du kan iterera igenom uppgiftslistan med hjälp av en `for..in`-slinga, men du måste använda operatorn för arrayåtkomst (`[]`) i stället för punktsyntax för att få åtkomst till värdet som associeras med respektive nyckel. Eftersom `masterTaskList` är en associativ array hämtas elementen inte nödvändigtvis i den ordning som du förväntar dig, vilket framgår av nästa exempel:

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
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
*/
```

Klona arrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I klassen Array finns det inga inbyggda metoder för att skapa kopior av en array. Du kan skapa en *ytligkopiering* av en array genom att anropa någon av metoderna `concat()` eller `slice()` utan argument. Om den ursprungliga arrayen har ett objekt i en ytlig kopiering, kopieras endast referenserna till objektet och inte själva objektet. Kopian pekar på samma objekt som den ursprungliga gör. Alla ändringar som görs på objektet återspeglas i båda arrayerna.

Vid en *djup kopiering* kopieras även alla objekt som hittas i den ursprungliga arrayen så att den nya arrayen inte pekar på samma objekt som den ursprungliga. För djup kopiering krävs mer än en kodrad där du vanligtvis gör ett anrop för att skapa en funktion. En sådan funktion kan skapas som ett globalt verktyg eller som en metod i en Array-underklass.

I följande exempel definieras en funktion med namnet `clone()`, som används för att göra djupkopieringen. Algoritmen lånas från en vanlig teknik som används vid Java-programmering. I funktionen skapas en djup kopia genom att serialisera arrayen i instanser av klassen ByteArray och sedan läsa tillbaka arrayen till en ny array. Funktionen accepterar ett objekt så att den kan användas både för indexerade arrayer och associativa arrayer, vilket framgår av följande exempel:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

Utökning av klassen Array

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Array är en av de få huvudklasserna som inte är slutgiltig, vilket innebär att du kan skapa egna underklasser till klassen Array. I detta avsnitt finns exempel på hur du skapar en underklass till klassen Array. Dessutom beskrivs något av vad som kan hända under processens gång.

Som nämnts tidigare är arrayer i ActionScript inte ”typade”, men du kan skapa en underklass av klassen Array som endast accepterar element med en viss datatyp. I exemplen i följande avsnitt definieras en Array-underklass med namnet TypedArray, där endast värden som innehåller datatypen som angavs i den första parametern tas emot. Klassen TypedArray visas bara som ett exempel på hur du kan utöka klassen Array och är av många anledningar inte lämplig i verkliga tillämpningar. För det första görs typkontrollen under körningen och inte vid kompileringen. För det andra ignoreras felmatchningen om den upptäcks i en TypedArray-metod och inget undantag genereras trots att metoden enkelt kan förändras så att detta görs. För det tredje kan klassen inte förhindra att arrayåtkomstoperatorn infogar värden med någon annan datatyp i arrayen. Slutligen favoriserar kodstilen enkelhet framför optimal prestanda.

Obs! *Du kan använda den teknik som beskrivs här när du skapar en typbestämd array. Det är emellertid bättre att använda ett Vector-objekt. En Vector-instans är en äkta typbestämd array, som har bättre prestanda och fler fördelar än klassen Array och underklasser. Syftet här är att visa hur du skapar en Array-underklass.*

Deklarera underklasser

Använd nyckelordet `extends` för att visa att en klass är en underklass till Array. I en Array-underklass ska attributet `dynamic` användas precis som i klassen Array. Annars fungerar underklassen inte som den ska.

I följande exempel visas definitionen av klassen TypedArray, som innehåller en konstant för datatypen, en konstruktormetod och fyra metoder för att lägga till element till arrayen. Koden för varje metod har utelämnats i detta exempel, men den beskrivs och förklaras i sin helhet i avsnitten som följer:

```
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 {}
}
```

I alla de fyra åsidosatta metoderna används AS3-namnutrymmet i stället för attributet `public`, eftersom vi i detta exempel antar att kompilatoralternativet `-as3` är `true` och att kompilatoralternativet `-es` är `false`. Detta är standardinställningarna för Adobe Flash Builder och AdobeFlashProfessional.

Om du är en van utvecklare, som föredrar att använda prototyparv, kan du göra två smärre förändringar i klassen TypedArray så att den kan kompileras med kompilatoralternativet `-es` som `true`. Börja med att ta bort alla förekomster av attributet `override` och ersätt AS3-namnutrymmet med attributet `public`. Därefter ersätter du alla fyra förekomster av `super` med `Array.prototype`.

Konstruktorn TypedArray

Underklasskonstruktorn utgör en intressant utmaning eftersom den måste kunna godta en argumentlista med godtycklig längd. Utmaningen är hur argumenten ska överföras till superkonstruktorn så att arrayen skapas. När du skickar en argumentlista som en array, betraktas den av superkonstruktorn som ett enskilt argument av typen Array och den resulterande arrayen kommer då alltid att vara ett element långt. Det traditionella sättet att hantera överförda argumentlistor är att använda metoden `Function.apply()`, som tar en array med argument som dess andra parameter men konverterar den till en argumentlista när funktionen körs. Tyvärr går det inte att använda metoden `Function.apply()` tillsammans med en konstruktor.

Det enda som återstår att göra är att återskapa logiken i Array-konstruktorn i konstruktorn TypedArray. I nästa exempel visas algoritmen som används i klassen Arrays konstruktor, som sedan återanvänds i din Array-underklasskonstruktor:

```
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]
            }
        }
    }
}
```

Koden i konstruktorn TypedArray har mycket gemensamt med koden för Array-konstruktorn, men fyra ändringar måste göras. Till att börja med innehåller parameterlistan en ny obligatorisk parameter med typen Class, som gör att det går att ange arrayens datatyp. För det andra är datatypen som skickas till konstruktorn tilldelad till variabeln `dataType`. För det tredje är värdet för egenskapen `length` i `else`-satsen tilldelat efter `for`-slingan så att `length` endast kommer att innehålla argument med rätt typ. Slutligen används i texten i `for`-slingan den version av metoden `push()` som åsidosätter argument så att endast de med rätt datatyp läggs till i arrayen. I följande exempel visas konstruktorfunktionen TypedArray:

```
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++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

TypedArray – metoder för åsidosättande

Klassen TypedArray åsidosätter fyra av klassen Arrays metoder för att lägga till element till en array. I vart och ett av fallen används i den åsidosättande metoden en typkontroll som förhindrar att element med fel datatyp läggs till. Detta innebär att varje metod anropar den egna överordnade klassens version.

Metoden `push()` itererar igenom argumentlistan med en `for..in`-slinga och en typkontroll görs av varje argument. De argument som inte har rätt typ tas bort från arrayen `args` med metoden `splice()`. När `for..in`-slingan är slut innehåller arrayen `args` endast värden med typen `dataType`. Den överordnade klassens version av `push()` anropas sedan med den uppdaterade versionen av arrayen `args`, vilket visas i följande exempel:

```
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));
}
```

Med metoden `concat()` skapas en temporär TypedArray med namnet `passArgs`. Den används för att lagra argument som har accepterats i typkontrollen. Detta medger att det går att återanvända koden för typkontrollen i metoden `push()`. Med en `for..in`-slinga itererar du genom `args`-arrayen och anropar `push()` vid varje argument. Eftersom `passArgs` har typen TypedArray körs TypedArray-versionen av `push()`. Metoden `concat()` anropar sedan den egna överordnade klassens version, vilket framgår i följande exempel:

```
    AS3 override function concat(...args):Array
    {
        var passArgs:TypedArray = new TypedArray(dataType);
        for (var i:* in args)
        {
            // type check done in push()
            passArgs.push(args[i]);
        }
        return (super.concat.apply(this, passArgs));
    }
```

Metoden `splice()` tar en godtycklig argumentlista, men de två första argumenten refererar alltid till ett indexvärde och antalet element som ska tas bort. Detta är orsaken till att det görs en typkontroll av elementen i arraken `args` endast vid indexposition 2 och högre i den åsidosatta metoden `splice()`. En intressant aspekt av koden är att det verkar finnas ett rekursivt anrop till `splice()` i `for`-slingan, men detta är trots allt inget rekursivt anrop eftersom `args` har type Array och inte TypedArray. Detta innebär att anropet `args.splice()` är ett anrop till den överordnade klassens version av metoden. När `for..in`-slingan är klar innehåller arrayen `args` endast värden med rätt typ i indexpositionerna 2 eller högre och `splice()`-anropet är till den egna överordnade klassens version, vilket visas i följande exempel:

```
    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));
    }
```

För metoden `unshift()`, där element läggs till i början av en array, accepteras också en godtycklig argumentlista. I den åsidosatta metoden `unshift()` används en algoritm som mycket påminner om den som används i metoden `push()`, vilket visas i följande exempel:

```
    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));
    }
}
```

Exempel på array: PlayList

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I exemplet PlayList visas olika tekniker för hur du arbetar med arrayer. Du lär dig samtidigt som du skapar ett program för att hantera en spellista med musiklåtar. Detta omfattar följande:

- Skapa en indexerad array
- Lägga till objekt i en indexerad array
- Sortera en array med objekt som har olika egenskaper genom att använda olika sorteringsalternativ
- Konvertera en array till en teckenavgränsad sträng

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Du hittar programfiler för PlayList i mappen Samples/PlayList. Programmet består av följande filer:

Fil	Beskrivning
PlayList.mxml eller PlayList.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/playlist/PlayList.as	En klass som representerar en lista med låtar. Klassen använder en array för att lagra listan, och hanterar sorteringen av objekten i listan.
com/example/programmingas3/playlist/Song.as	Ett värdeobjekt som representerar information om en låt. Objekten som hanteras i klassen PlayList är Song-instanser.
com/example/programmingas3/playlist/SortProperty.as	En pseudonumrering vars tillgängliga värden representerar egenskaper i klassen Song där en lista med Song-objekt kan sorteras.

klassen PlayList – översikt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen PlayList hanterar du en uppsättning Song-objekt. Den har publika metoder med funktioner för att lägga till en låt i spellistan (metoden `addSong()`) och för att sortera låtar i listan (metoden `sortList()`). Dessutom innehåller klassen en skrivskyddad åtkomstegenskap, `songList`, som ger dig tillgång till de faktiska låtarna i spellistan. Internt fungerar det så att klassen PlayList håller ordning på låtarna i en privat Array-variabel:

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

Förutom Array-variabeln `_songs`, som används i klassen PlayList för att hålla ordning på dess lista med låtar, finns två andra privata variabler för att dels hålla ordning på om listan behöver sorteras (`_needToSort`) och dels vilken egenskap som ska användas för att sortera låtlistan vid en given tidpunkt (`_currentSort`).

Precis som för alla objekt är deklarationen av en Array-instans endast halva jobbet när du skapar en Array. Innan du får åtkomst till egenskaper eller metoder för en Array-instans måste den skapas, vilket görs i klassen PlayLists konstruktor.

```
public function PlayList()
{
    this._songs = new Array();
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
}
```

Den första raden i konstruktorn instansierar variabeln `_songs` så att den är klar att användas. Dessutom anropas metoden `sortList()` för att ange den första sorteringsegenskapen.

Lägga till en låt i listan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När en användare skriver in en ny låt i programmet används koden i inmatningsformuläret för att anropa metoden `addSong()` i klassen PlayList.

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

I `addSong()` anropas metoden `push()` för arrayen `_songs`, och objektet Song, som skickades till `addSong()`, läggs till som ett nytt element i den arrayen. Med metoden `push()` läggs det nya elementet till i slutet av arrayen, oberoende av sortering som kan ha gjorts tidigare. Detta betyder, att sedan metoden `push()` har anropats är det troligtvis så att spellistan inte längre är korrekt sorterad. Därför sätts variabeln `_needToSort` till `true`. Teoretiskt skulle det nu gå att omedelbart anropa metoden `sortList()` och undanröja behovet av att hålla reda på om listan är sorterad eller inte vid en viss tidpunkt. I praktiken finns det emellertid inget behov av att sortera spellistan förrän omedelbart innan den hämtas. Genom att skjuta upp sorteringen kommer ingen onödig sortering att ske när till exempel flera låtar läggs till innan listan hämtas.

Sortera en lista med låtar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom Song-instanserna som hanteras med spellistan är komplicerade objekt, kan användare av programmet vilja använda olika egenskaper för att sortera den, till exempel titel och år. I programmet PlayList innehåller uppgiften att sortera listan tre delar: identifiera egenskapen som ska användas för sorteringen, ange vilka sorteringsalternativ som ska användas för att sortera efter den egenskapen och utföra den faktiska sorteringsoperationen.

Sorteringsegenskaper

Ett Song-objekt används för att hålla reda på flera egenskaper, inklusive titel, artist, år, filnamn och användarens genreval. Av dessa lämpar sig endast de första tre för sortering. För att underlätta för utvecklarna innehåller exemplet klassen SortProperty, som fungerar som en numrering med värden som representerar de egenskaper som är tillgängliga för sortering.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

Klassen SortProperty innehåller tre konstanter, `TITLE`, `ARTIST` och `YEAR`, vilka alla lagrar en sträng med det faktiska namnet på den associerade egenskap för klassen Song som kan användas för sorteringen. Närhelst en sorteringsegenskap anges i resten av koden görs det med numreringsmedlemmen. I exempelvis konstruktorn PlayList sorteras listan till att börja med genom att metoden `sortList()` anropas enligt följande:

```
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
```

Eftersom sorteringsegenskapen anges som `SortProperty.TITLE` sorteras låtarna efter deras titlar.

Sortera efter egenskap och ange sorteringsalternativ

Den faktiska sorteringen av spellistan görs med klassen PlayList i metoden `sortList()` enligt följande:

```
/**
 * Sorts the list of songs according to the specified property.
 */
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;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

När sorteringen sker efter titel eller artist passar det bäst att göra den i bokstavsordning, men när den sker efter årtal är det mest logiska att göra en numerisk sortering. Satsen `switch` används för att definiera rätt sorteringsalternativ, som sparas i variabeln `sortOptions`, enligt det värde som angetts i parametern `sortProperty`. Här används återigen de namngivna numreringsmedlemmarna, i stället för de hårdkodade värdena, för att skilja egenskaperna åt.

När sorteringsegenskapen och sorteringsalternativen är bestämda sorteras arrayen `_songs` genom att metoden `sortOn()` anropas, med dessa två värden som parametrar. Den aktuella sorteringsegenskapen spelas in och den aktuella spellistan sorteras.

Kombinera arrayelement i en teckenavgränsad sträng

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Förutom att använda en array för att underhålla spellistan i klassen PlayList används i detta exempel arrayer även i klassen Song för att hantera genrelistan som låten tillhör. Titta på följande kodutdrag från definitionen för klassen Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

När du skapar en ny Song-instans definieras `genres`-parametern, som används för att ange vilken genre som låten tillhör, som en Array-instans. Detta är praktiskt när du vill gruppera flera genrer i en enda variabel som kan skickas till konstruktorn. Internt behåller emellertid klassen Song genrerna i den privata variabeln `_genres` som en semikolonavgränsad String-instans. Array-parametern konverteras till en semikolonavgränsad sträng genom att dess `join()`-metod anropas med stränglitteralvärdet `";"` som den angivna avgränsaren.

På samma sätt tillåter `genres`-åtkomster att genrer ställs in eller hämtas som en array:

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

Åtkomsten `genresset` uppträder på exakt samma sätt som konstruktorn; den accepterar en array och anropar metoden `join()` för att konvertera den till en semikolonavgränsad sträng. Med åtkomsten `get` utförs motsatt åtgärd: `_genres`-variabelns `split()`-metod anropas och strängen delas upp i en värdearray med hjälp av den angivna avgränsaren (stränglitteralvärdet `";"` som tidigare).

Kapitel 4: Felhantering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att hantera fel innebär att skapa en logik för hur programmet ska svara på, eller åtgärda, ett fel. Fel uppstår antingen när ett program kompileras eller när ett kompilerat program körs. Om programmet hanterar fel händer *någonting* som svar på ett inträffat fel, i motsats till inget svar (när ett fel uppstår utan att det märks). Om felhanteringen används på rätt sätt skyddas programmet och användarna slipper oväntade beteenden.

Felhanteringen är emellertid ett omfattande ämne, som behandlar många olika typer av fel som kan inträffa under kompilering eller när ett program körs. Här beskrivs närmare hur du hanterar körningsfel (fel som inträffar när ett program körs), de olika typer av fel som kan uppstå och fördelarna med felhanteringen i ActionScript 3.0.

Felhanteringens grunder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett körtidsfel är något som går snett i ActionScript-koden och hindrar ActionScript-innehållet från att köras som det är tänkt. För att vara säker på att ActionScript-koden kan köras av alla användare ska du skriva koden i det program som hanterar felet, åtgärdar eller går runt det eller meddelar användaren om det. Denna process kallas *felhantering*.

Felhanteringen är ett omfattande ämne, som behandlar många olika typer av fel som kan inträffa under kompilering eller när ett program körs. Fel som inträffar vid kompileringen är ofta lättare att identifiera. Åtgärda dem så att SWF-filen kan skapas.

Körningsfel kan vara svåra att upptäcka eftersom den felaktiga koden måste köras för att felet ska upptäckas. Om ett segment i programmet har flera kodgrenar, som en `if..then..else`-sats testar du alla möjliga villkor, med alla möjliga indatavärden som verkliga användare kan tänkas använda, så att du kan vara säker på att koden är felfri.

Körningsfel kan delas upp i två kategorier: *programfel* är misstag i ActionScript-koden, t.ex. angivande av fel datatyp för en metodparameter; *logiska fel* är misstag i programmeringslogiken (datakontroll och värdeförändringar), t.ex. användning av fel formel för att beräkna räntor i en banktillämpning. Som nämnts tidigare kan båda dessa typer av fel upptäckas och rättas till i förväg genom en omsorgsfull testning av programmet.

Du vill naturligtvis identifiera och ta bort alla fel från programmet innan det släpps till slutanvändarna. Det finns emellertid fel som inte går att förutse eller förhindras. Anta att ditt ActionScript-program läser in information från en speciell webbplats som du inte kan kontrollera. Om då webbplatsen vid någon tidpunkt inte är tillgänglig så kommer den del av programmet som är beroende av dessa externa data inte att fungera korrekt. Den viktigaste aspekten av felhantering är hur du förbereder för dessa oplanerade situationer och hur du hanterar dem på ett bra sätt. Användarna måste kunna fortsätta att använda programmet eller åtminstone få ett vänligt felmeddelandet om varför det inte fungerar.

Körningsfel kan uppdelas i två kategorier i ActionScript:

- Felklasser: Många fel tillhör en felklass. När ett fel inträffar skapas (i till exempel Flash Player eller Adobe AIR) en instans av den specifika felklassen som är associerad med det aktuella felet. Du kan i koden använda den information som finns i felobjektet för att hantera felet på lämpligt sätt.

- Felhändelser: Ibland inträffar ett fel när i vanliga fall en händelse borde ha utlösts i Flash-miljön. I så fall utlöses en felhändelse i stället. Varje felhändelse är associerad till en klass, och i Flash skickas en instans av den klassen till de metoder som prenumererar på felhändelsen.

För att avgöra om en viss metod kan utlösa ett fel eller en felhändelse läser du om metoden i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer för programmering av felhanteringsrutiner:

Asynkron Ett programkommando, t.ex. ett metodanrop, som inte ger ett omedelbart resultat. Det resulterar i stället i ett resultat (eller ett fel) i form av en händelse.

Fånga upp När ett undantag (ett körningsfel) inträffar och koden upptäcker undantaget heter det att koden *fångar upp* undantaget. När ett undantag väl har fångats upp skickas inga fler meddelande om undantaget till annan ActionScript-kod.

Felsökningsversion En särskild version av Flash-miljön, till exempel felsökningsversionen av Flash Player eller AIR Debug Launcher (ADL), som innehåller kod för att informera användaren om körningsfel. I standardversionen av Flash Player och Adobe AIR (den som de flesta användarna har) ignoreras fel som inte hanteras i ActionScript-koden. I felsökningsversionerna (som finns i Adobe Flash CS4 Professional och Adobe Flash Builder) visas ett varningsmeddelande om ett ohanterat fel inträffar.

Undantag Ett fel som inträffar när ett program körs och som Flash-miljön inte kan lösa.

Återkasta När koden fångar upp ett undantag skickas inga fler meddelande om undantaget till andra objekt. Om det har betydelse för andra objekt att undantaget har inträffat måste koden *återkasta* undantaget för att starta meddelandeprocessen på nytt.

Synkron Ett programkommando, till exempel ett metodanrop, som ger ett omedelbart resultat (eller omedelbart ger ett fel) så att svaret kan användas i samma kodblock.

Ge Att meddela Flash-miljön (och följaktligen meddela andra objekt och annan ActionScript-kod) att ett fel har inträffat kallas för att *ge* ett fel.

Olika typer av fel

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du utvecklar och kör program kommer du att upptäcka olika typer av fel och felterminologi. I listan beskrivs de vanligaste feltyperna och begreppen:

- *Kompileringsfel inträffar när ett fel upptäcks under kompileringen i ActionScript-kompilatorn.* Kompileringsfel visas när syntaktiska problem i koden förhindrar att programmet skapas.
- *Körningsfel inträffar när du kör programmet efter kompileringen.* Körningsfel är fel som inträffar när en SWF-fil spelas i Flash-miljön (till exempel Adobe Flash Player eller Adobe AIR). I de flesta fall hanterar du körningsfel när de inträffar, rapporterar dem till användaren och vidtar de åtgärder som krävs för att programmet ska kunna fortsätta att köras. Om ett allvarligt fel inträffar, om det till exempel inte går att ansluta till en fjärrwebbplats eller läsa in nödvändiga data, kan du använda felhantering för att se till att programmet stängs på ett bra sätt.
- *Synkrona fel* är körningsfel som inträffar när en funktion anropas, till exempel om du använder en viss metod och det argument som skickas till metoden är ogiltigt och således genererar ett undantag. De flesta felen är synkrona, när programsatsen körs, och flödeskontrollen överförs omedelbart till den lämpligaste `catch`-satsen.

Följande exempel kommer att resultera i ett körningsfel eftersom metoden `browse()` inte anropas innan ett försök görs att hämta en fil:

```
var fileRef:FileReference = new FileReference();
try
{
    fileRef.upload(new URLRequest("http://www.yourdomain.com/fileupload.cfm"));
}
catch (error:IllegalOperationError)
{
    trace(error);
    // Error #2037: Functions called in incorrect sequence, or earlier
    // call was unsuccessful.
}
```

I detta fall inträffar ett körningsfel synkront eftersom det i Flash Player upptäcks att metoden `browse()` inte anropades innan hämtningsförsöket gjordes.

Mer information om synkron felhantering finns i ”Hantera synkrona fel i ett program” på sidan 56.

- *Asynkrona fel* är körningsfel som inträffar utanför det normala programflödet. De skapar händelser och händelseavlyssnare fångar upp dem. En asynkron åtgärd är en åtgärd där en funktion initierar en åtgärd, men inte väntar på att den ska avslutas. Du kan skapa en felhändelseavlyssnare för att vänta på att programmet eller användaren ska utföra någon åtgärd. Om åtgärden misslyckas fångas felet upp av händelseavlyssnaren och felhändelsen besvaras. Därefter anropar händelseavlyssnaren en händelsehanterarfunktion för att besvara felhändelsen på ett verkningsfullt sätt. Händelsehanteraren kan öppna en dialogruta med information om hur användaren ska åtgärda det uppkomna felet.

  Om du som i exemplet tidigare med det synkrona felet vid filhämtning. anropar metoden `browse()` innan du startar hämtningen kommer Flash Player att sända ut flera händelser. När exempelvis en hämtning startas skickas händelsen `open`. När hämtningsoperationen är avslutad skickas händelsen `complete`. Eftersom händelsehantering är asynkron (d.v.s. inte inträffar vid en speciell, känd och i förväg bestämd tidpunkt), ska du använda metoden `addEventListener()` för att avlyssna dessa specifika händelser, som exemplet nedan visar:

```
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);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

Mer information om asynkron felhantering finns i ”Svara på felhändelser och status” på sidan 62.

- *Ej infångade undantag är fel som orsakas av att ingen motsvarande logik (som en catch-sats) besvarar felet.* Om ett fel inträffar i programmet och ingen lämplig `catch`-sats eller händelsehanterare som kan hantera felet hittas på den aktuella nivån eller en högre nivå, kommer felet att betraktas som ett ej infångat undantag.

  När ett fel som inte infångades inträffar skickas en `uncaughtError`-händelse. Den här händelsen kallas också för ”global felhanterare”. Händelsen skickas av SWF-filens UncaughtErrorEvents-objekt, som är tillgängligt via egenskapen `LoaderInfo.uncaughtErrorEvents`. Om inga avlyssnare har registrerats för `uncaughtError`-händelsen, ignorerar körningen dessa fel och försöker att fortsätta, så länge som felet inte stoppar SWF:en.

  Förutom att skicka händelsen `uncaughtError` svarar Flash-körningens felsökningsversioner på fel som inte fångats in genom att avbryta det aktuella skriptet. Felet som inte fångats in visas sedan i en `trace`-sats eller i ett felmeddelandet i en loggfil. Om undantagsobjektet är en instans av Error-klassen eller dess underklasser visas även stackspårningsinformation i utdata. Mer information om hur du använder de olika felsökningsversionerna av Flash finns i ”Arbeta med felsökningsversioner av Flash-miljöer” på sidan 56.

  Obs! *När en uncaughtError-händelse hanteras, och en error-händelse genereras från en uncaughtError-hanterare, anropas händelsehanteraren flera gånger. Detta ger upphov till en oändlig slinga med undantagsfel. Du bör undvika ett sådant scenario.*

Felhantering i ActionScript 3.0

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom många program kan köras utan någon logik för att hantera fel är det många utvecklare som försöker skjuta upp felhanteringen i sina program. Men utan någon form av felhantering är det lätt att programmet hänger sig eller att användaren blir frustrerad över att programmet inte fungerar som det ska. I ActionScript 2.0 finns en Error-klass som du kan använda för att skapa egna logiska funktioner så att ett specifikt meddelande ges när något oväntat inträffar. Eftersom felhantering är en viktig del av ett användarvänligt program, innehåller ActionScript 3.0 en utökad arkitektur för att fånga upp eventuella fel.

Obs! *I Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen beskrivs undantag som genereras av många metoder, men alla möjliga undantag för alla metoder finns kanske inte med. En metod kan generera ett undantag vid syntaxfel eller något annat problem som inte uttryckligen beskrivs i metodbeskrivningen, även om beskrivningen listar några av de undantag som metoden kan generera.*

Felhanteringselement i ActionScript 3.0

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 finns många verktyg för felhantering, bland annat:

- Felklasser. ActionScript 3.0 innehåller en mängd Error-klasser som täcker in fler situationer som kan generera felobjekt. Varje felklass används i programmet för att hantera och besvara specifika felförhållanden, oberoende av om de är relaterade till systemfel (som ett MemoryError-villkor), kodningsfel (som ett ArgumentError-villkor), nätverks- eller kommunikationsfel (som ett URIError-villkor) eller andra situationer. Mer information om varje klass finns i ”Jämföra felklasserna” på sidan 65.
- Färre tysta fel. I tidigare versioner av Flash Player genererades och rapporterades fel bara när du använde programsatsen `throw`. I Flash Player 9 och senare, genererar interna ActionScript-metoder och -egenskaper körningsfel. Dessa fel gör det möjligt att hantera undantagen mer effektivt när de inträffar, i stället för att agera på varje undantag individuellt.

- Tydliga felmeddelanden visas under felsökningen. När du använder felsökningsversionen av en Flash-miljö genererar problematiska kodavsnitt eller situationer tydliga felmeddelanden, som hjälper dig att identifiera orsaken till varför ett visst kodblock inte fungerar. Tack vare dessa meddelanden kan du åtgärda felen mer effektivt. Du hittar mer information i ”Arbeta med felsökningsversioner av Flash-miljöer” på sidan 56.
- Exakta fel innebär tydligare felmeddelanden för användaren. I tidigare versioner av Flash Player returnerade metoden `FileReference.upload()` det booleska värdet `false` om `upload()`-anropet inte lyckades. Detta visade att det fanns fem möjliga fel. Om ett fel inträffar när du anropar metoden `upload()` i ActionScript 3.0, kan fyra olika fel hjälpa dig att visa mer exakta felmeddelanden för slutanvändarna.
- Förfinad felhantering. Distinkta fel erhålls för många vanliga situationer. I ActionScript 2.0 hade exempelvis egenskapen `name` värdet `null` innan ett FileReference-objekt fylldes i med värden (vilket innebar att du, innan du kunde använda eller visa egenskapen `name`, var tvungen att se till att ett värde angavs, och inte `null`). Om du i ActionScript 3.0 försöker få åtkomst till egenskapen `name` innan den har ett värde, kommer felet IllegalOperationError att levereras från Flash Player eller AIR, vilket anger att värdet inte är angivet och måste du sedan använda `try..catch..finally`-blocken för att hantera felet. Mer information finns i ”Använda try..catch..finally-satser” på sidan 56.
- Inga märkbara prestandaförsämringar. När du använder `try..catch..finally`-blocken för felhantering kommer små eller inga ytterligare resurser att tas i anspråk om du jämför med tidigare versioner av ActionScript.
- Du använder en ErrorEvent-klass för att bygga upp avlyssnare för specifika asynkrona felhändelser. Mer information finns i ”Svara på felhändelser och status” på sidan 62.

Strategier för felhantering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Så länge inga fel uppstår i programmet kan det fortsätta att fungera utan att du bygger in någon logik för felhantering i koden. Om du emellertid väljer att inte aktivt hantera fel och ett problem uppstår, kommer användarna aldrig att förstå varför programmet inte gör som det är tänkt.

Det finns olika sätt att närma sig felhanteringsproblematiken i programmet. Här nedan sammanfattas de tre vanligaste sätten att hantera fel:

- Använda `try..catch..finally`-satser. Dessa satser fångar upp synkrona fel när de inträffar. Du kan kapsla programsatserna i en hierarkisk struktur för att fånga upp undantagen på olika nivåer när koden körs. Mer information finns i ”Använda try..catch..finally-satser” på sidan 56.
- Skapa egna felobjekt. Du kan använda klassen Error för att skapa egna felobjekt för att spåra operationer i programmet som inte täcks av de byggda feltyperna. Sedan kan du använda `try..catch..finally`-satser i egna felobjekt. Mer information finns i ”Skapa egna felklasser” på sidan 61.
- Skriva händelseavlyssnare och hanterare för att besvara felhändelser. Om du väljer den här strategin kan du skapa en global felhantering där liknande händelser hanteras utan att du behöver duplicera en massa kod i `try..catch..finally`-block. Det troligaste är att du med det här sättet att arbeta kommer att fånga upp asynkrona fel. Mer information finns i ”Svara på felhändelser och status” på sidan 62.

Arbeta med felsökningsversioner av Flash-miljöer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Adobe tillhandahåller särskilda versioner av Flash-miljön för utvecklare för att hjälpa dem med felsökning. Du får en kopia av felsökningsversionen för Flash Player när du installerar Adobe Flash Professional eller Adobe Flash Builder. Du får också ett verktyg för att felsöka Adobe AIR-program, som kallas för ADL, när du installerar något av dessa verktyg eller som en del av Adobe AIR SDK.

Det finns väsentliga skillnader mellan hur felsökningsversionerna och releaseversionerna av Flash Player och Adobe AIR uppvisar fel. I felsökningsversionerna visas feltypen (t.ex. ett allmänt Error, IOError eller EOFError), felnummer och textbaserade felmeddelanden. I releaseversionerna visas endast feltypen och felnumret. Titta på följande kod:

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

Om metoden `readBoolean()` resulterar i ett EOFError i felsökningsversionen av Flash Player, visas följande meddelande i textfältet `tf`: "EOFError: Error #2030: End of file was encountered."

Samma kod i en officiell version av Flash Player eller Adobe AIR skulle visa följande text: ”EOFError: Error #2030.”

Obs! *Felsökningsspelarna sänder en händelse med namnet "allComplete". Undvik därför att skapa anpassade händelser med namnet ”allComplete”. Annars kan oförutsägbara beteenden uppstå vid felsökningen.*

För att minimera resurserna och storleken på releaseversionerna finns inga felmeddelandesträngar med. Du kan slå upp felnumret i dokumentationen (bilagorna till Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen) och få hjälp med att tolka ett felmeddelande. Du kan även välja att återskapa felet i felsökningsversionerna av Flash Player och AIR för att se det fullständiga meddelandet.

Hantera synkrona fel i ett program

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Synkron felhanteringslogik är den vanligaste felhanteringstekniken. Här infogar du programsatser i koden för att fånga upp synkrona fel som inträffar när ett program körs. Den här typen av felhantering gör att du i programmet kan uppmärksamma och åtgärda körningsfel när funktioner inte fungerar. Logiken i att fånga upp ett synkront fel innehåller `try..catch..finally`-satser, vilka bokstavligen prövar en åtgärd, fångar upp ett svar från Flash-miljön och slutligen utför andra åtgärder för att hantera den misslyckade åtgärden.

Använda try..catch..finally-satser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du arbetar med synkrona körningsfel ska du använda `try..catch..finally`-satser för att fånga upp fel. När ett körningsfel inträffar genererar Flash-miljön ett undantag, vilket betyder att programmet upphör att köras normalt och ett specialobjekt av typen Error skapas. Objektet Error skickas sedan till det första tillgängliga `catch`-blocket.

Programsatsen `try` omsluter programsatser där fel kan uppstå. Du använder alltid programsatsen `catch` med en `try`-sats. Om ett fel upptäcks i någon av programsatserna i programsatsblocket `try` kommer den `catch`-sats som är kopplad till programsatsen `try` att köras.

Programsatsen `finally` omfattar programsatser som körs om ett fel inträffar i blocket `try`. Om det inte finns något fel kommer de programsatser som finns i blocket `finally` att köras när blockprogramsatsen `try` är avslutad. Om det finns ett fel kommer rätt `catch`-sats att köras först, följd av programsatserna i blocket `finally`.

Med följande kod demonstreras syntaxen som används för `try..catch..finally`-satser:

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Varje `catch`-sats identifierar en specifik typ av undantag som den hanterar. Programsatsen `catch` kan endast ange felklasser som är underklasser till klassen Error. Varje `catch`-sats kontrolleras i rätt ordning. Endast den första `catch`-satsen som överensstämmer med det aktuella felet körs. Med andra ord, om du först kontrollerar Error-klassen på en högre nivå och sedan en underklass till Error-klassen kommer endast den på den högre nivån att överensstämma. I följande exempel illustreras detta:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

Koden resulterar i följande utdata:

```
<Error> I am an ArgumentError
```

Om du vill kunna fånga upp ArgumentError ska du se till att det mest specifika felet listas först och att mer allmänna feltyper listas senare, vilket visas i nästa exempel:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

Flera metoder och egenskaper i API:t för ActionScript resulterar i körningsfel om ett fel inträffar när de körs. Metoden `close()` i klassen Sound resulterar exempelvis i ett IOError-fel om metoden inte kan stänga ljudströmmen. Detta visas i nästa exempel:

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

Efterhand som du blir mer bekant med Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen kommer du att lära dig vilka metoder som genererar undantag, vilket beskrivs i avsnitten om de olika metoderna.

Programsatsen throw

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash-miljöerna genererar undantag när de stöter på fel i det program som körs. Du kan dessutom skapa egna undantag genom att använda programsatsen `throw`. När du vill skapa egna fel rekommenderar Adobe att du använder instanser av klassen Error eller dess underklass. I följande kod visas en `throw`-sats som returnerar en instans av klassen Error, `MyErr`, och eventuellt anropar en funktion, `myFunction()`, för att besvara felet:

```
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; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

Observera att programsatsen `catch` är uppställd så att de mest signifikanta datatyperna är listade först. Om `catch`-satsen för Number-datatypen sätts först kommer varken catch-satsen för uint-datatypen eller catch-satsen för int-datatypen att köras.

Obs! *I Java-programmering måste alla funktioner som kan resultera i ett undantag deklarera detta och visa de undantagsklasser som de kan ge upphov till i en `throws`-sats som är kopplad till funktionsdeklarationen. I ActionScript behöver du inte deklarera de undantag som kan orsakas av en funktion.*

Visa enkla felmeddelanden

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En av de största fördelarna med den nya undantags- och felmodellen är att den ger dig möjlighet att meddela användarna när och varför en åtgärd misslyckades. Din del blir att skriva koden för att visa meddelandet och erbjuda alternativa lösningar.

I följande exempel visas en enkel `try..catch`-sats som visar ett fel i ett textfält:

```
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);
            }
        }
    }
}
```

Tack vare ett stort antal felklasser och inbyggda kompileringsfel erbjuder ActionScript 3.0 mer information än tidigare versioner av ActionScript om varför ett fel har uppstått. Med den här informationen kan du skapa mer stabila program med bättre felhantering.

Skicka tillbaka fel

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar ett program kan det finnas situationer där du måste skicka tillbaka ett fel eftersom du inte kan hantera felet på ett tillfredsställande sätt. I nästa exempel visas ett kapslat `try..catch`-block, där ett anpassat ApplicationError skickas tillbaka om felet inte kan hanteras i det kapslade `catch`-blocket:

```
try
{
    try
    {
        trace("<< try >>");
        throw new ApplicationError("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);
}
```

Resultatet av det föregående kodutdraget skulle kunna se ut så här:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

Det kapslade `try`-blocket ger ett anpassat ApplicationError-fel som fångas upp i det efterföljande `catch`-blocket. I det kapslade `catch`-blocket kan felet hanteras och om det lyckas kommer ett ApplicationError-objekt att skickas till det omslutande `try..catch`-block.

Skapa egna felklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan i ActionScript förändra en av standardfelklasserna och skapa egna felklasser. Det kan finnas flera orsaker till varför du ska skapa egna felklasser:

- När du vill identifiera specifika fel eller grupper med fel som är unika för programmet.

  Vidta till exempel andra åtgärder för fel som genereras av din egen kod, utöver de som fångas upp av Flash-miljön. Du skapar då en underklass till klassen Error som spårar den nya feldatatypen i `try..catch`-blocken.

- När du vill visa unika felmeddelanden för fel som uppkommer i programmet.

  Du kan skapa en ny `toString()`-metod för att formatera dina felmeddelanden på ett speciellt sätt. Du kan också definiera en `lookupErrorString()`-metod som tar en felkod och hämtar rätt meddelande beroende på användarens språkinställningar.

En anpassad felklass måste utöka det väsentliga i ActionScript-felklassen. Här visas ett exempel på en anpassad AppError-klass som utökar klassen Error:

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

I följande exempel visas hur du kan använda AppError i dina projekt:

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

Obs! *Om du vill åsidosätta metoden `Error.toString()` i din underklass, ger du den en `...`-parameter (rest). ECMAScript-språkspecifikationen som ActionScript 3.0 baseras på definierar metoden `Error.toString()` på det sättet, och ActionScript 3.0 definierar den på samma sätt för att ge bakåtkompatibilitet. Parametrarna måste därför överensstämma exakt när du åsidosätter metoden `Error.toString()`. Du ska inte skicka några parametrar till metoden `toString()` under körningen eftersom dessa parametrar kommer att ignoreras.*

Svara på felhändelser och status

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En av de mest märkbara förbättringarna när det gäller felhantering i ActionScript 3.0 är stödet för felhändelsehantering för att svara på asynkrona fel som inträffar när ett program körs. (En definition av asynkrona fel finns i ”Olika typer av fel” på sidan 52.)

Du kan skapa händelseavlyssnare och händelsehanterare för att besvara felhändelser. I många klasser skickas felhändelser på samma sätt som andra händelser. En instans av exempelvis klassen XMLSocket skickar normalt tre typer av händelser: `Event.CLOSE`, `Event.CONNECT` och `DataEvent.DATA`. Om emellertid ett problem inträffar kan klassen XMLSocket skicka `IOErrorEvent.IOError` eller `SecurityErrorEvent.SECURITY_ERROR`. Mer information om händelseavlyssnare och händelsehanterare finns i ”Hantera händelser” på sidan 125.

Felhändelser passar in i en av dessa två kategorier:

- Felhändelser som utökar klassen ErrorEvent

  Klassen flash.events.ErrorEvent innehåller egenskaper och metoder för att hantera fel som rör nätverks- och kommunikationsåtgärder i ett program som körs. Klasserna AsyncErrorEvent, IOErrorEvent och SecurityErrorEvent utökar klassen ErrorEvent. Om du använder felsökningsversionen av en Flash-miljö öppnas en dialogruta under körningen som informerar dig om felhändelser, utan avlyssnarfunktioner, som identifieras av spelaren.

- Statusbaserade felhändelser

Statusbaserade felhändelser är relaterade till egenskaperna `netStatus` och `status` i nätverks- och kommunikationsklasserna. Om en Flash-miljö stöter på ett problem när data läses eller skrivs anges värdet för egenskapen `netStatus.info.level` eller `status.level` (beroende på vilket klassobjekt du använder) till `error`. Du besvarar detta fel genom att kontrollera om egenskapen `level` innehåller värdet `"error"` i din händelsehanterarfunktion.

Arbeta med felhändelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen ErrorEvent och dess underklasser innehåller feltyper för hantering av fel som skickas av Flash-miljön vid skriv- och läsåtgärder.

I följande exempel används både en `try..catch`-sats och felhändelsehanterare för att visa fel som upptäcks när ett försök görs att läsa en lokal fil. Du kan på de platser som markeras med kommentaren ”your error handling code here” lägga till en mer sofistikerad kod för att ge användarna flera alternativ eller möjlighet att välja att hantera felet automatiskt:

```
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;
    import flash.text.TextFieldAutoSize;

    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);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

Arbeta med statusförändringshändelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash-miljöer ändrar värdet på egenskapen `netStatus.info.level` eller `status.level` dynamiskt för de klasser som har stöd för egenskapen `level` när ett program körs. Klasser med egenskapen `netStatus.info.level` är NetConnection, NetStream och SharedObject. De klasser som innehåller egenskapen `status.level` är HTTPStatusEvent, Camera, Microphone och LocalConnection. Du kan skriva en hanterarfunktion för att besvara ändringar i `level`-värdet och för att spåra kommunikationsfel.

I följande exempel används funktionen `netStatusHandler()` för att testa värdet för egenskapen `level`. Om egenskapen `level` indikerar att ett fel har inträffat kommer meddelandet ”Video stream failed” att spåras av koden.

```
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);
        }
    }
}
```

Jämföra felklasserna

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript finns ett antal fördefinierade felklasser. Men du kan också använda samma felklasser i din egen kod. Det finns två typer av felklasser i ActionScript 3.0: ActionScript-huvudfelklasser och felklasser i flash.error-paketet. I paketet flash.error finns ytterligare klasser som ska underlätta vid programutveckling och felsökning i ActionScript 3.0.

Huvudfelklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Huvudfelklasserna omfattar klasserna Error, ArgumentError, EvalError, RangeError, ReferenceError, SecurityError, SyntaxError, TypeError, URIError och VerifyError. Dessa klasser finns representerade på namnutrymmets översta nivå.

Klassnamn	Beskrivning	Anteckningar
Error	Klassen Error används för att hantera undantag och är basklass för andra undantagsklasser som definieras i ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError och URIError.	Klassen Error utgör basklassen för alla körningsfel och är den basklass som rekommenderas för anpassade felklasser.
ArgumentError	Klassen ArgumentError representerar ett fel som uppstår när parametervärden som anges i en funktion inte matchar parametrarna som är definierade för funktionen.	Följande typer av argumentfel kan förekomma: • För få eller för många argument skickas till en metod. • Ett argument förväntades vara en numreringsmedlem, men var det inte.
EvalError	Ett EvalError-undantag skapas om en parameter skickas till konstruktorn i klassen Function eller om funktionen `eval()` anropas i användarkoden.	I ActionScript 3.0 har stödet för funktionen `eval()` tagits bort och försök att använda funktionen resulterar i ett fel. I tidigare versioner av Flash Player användes funktionen `eval()` för att få åtkomst till variabler, egenskaper, objekt eller filmklipp.
RangeError	Ett RangeError-undantag genereras om ett numeriskt värde ligger utanför det tillåtna intervallet.	Till exempel genereras ett RangeError-undantag från klassen Timer om en fördröjning var negativ eller utan gräns. Du kommer också att erhålla ett RangeError-undantag om du försöker lägga till ett visningsobjekt med ett ogiltigt djup.
ReferenceError	Ett ReferenceError-undantag returneras när ett försök att referera till en odefinierad egenskap görs på ett fast (icke-dynamiskt) objekt. De versioner av ActionScript-kompilatorn som fanns före ActionScript 3.0 gav inga fel när ett försök gjordes att få åtkomst till en egenskap som var `undefined`. ActionScript 3.0 returnerar emellertid ReferenceError-undantaget i detta fall.	Undantag för odefinierade variabler kan innebära att det finns fel och de hjälper dig att förbättra programkvaliteten. Men om du inte är van vid att initiera variablerna kan det här nya kännas ovant och du måste kanske göra vissa förändringar i ditt sätt att koda.
SecurityError	Ett SecurityError-undantag erhålls när en säkerhetsöverträdelse äger rum och åtkomsten nekas.	Följande typer av säkerhetsfel kan förekomma: • En otillåten egenskapsåtkomst eller ett otillåtet metodanrop görs utanför säkerhetssandlådans gräns. • Ett försök att komma åt en URL-adress som inte är tillåten av säkerhetssandlådan gjordes. • Ett försök till socketanslutning gjordes på en port men den nödvändiga socketprincipfilen fanns inte på plats. • Ett försök gjordes att få åtkomst till användarens kamera eller mikrofon men användaren nekade åtkomst till enheten.
SyntaxError	Det uppstår ett SyntaxError-undantag när ett tolkningsfel sker i ActionScript-koden.	Ett SyntaxError-fel kan genereras under följande omständigheter: • ActionScript genererar SyntaxError-undantag om RegExp-klassen tolkar ett ogiltigt reguljärt uttryck. • ActionScript genererar SyntaxError-undantag om XMLDocument-klassen tolkar ogiltig XML.

Klassnamn	Beskrivning	Anteckningar
TypeError	Det uppstår ett TypeError-undantag när den faktiska typen av operand skiljer sig från den förväntade typen.	Du kan få ett TypeError-undantag vid följande tillfällen: • När en funktionsparameter eller metod inte kan tvingas till den formella parametertypen. • När ett värde tilldelas till en variabel och inte kan tvingas till variabeltypen. • Höger sida om operatorn `is` eller `instanceof` inte är en giltig typ. • Nyckelordet `super` används felaktigt. • En egenskapssökning resulterar i fler än en bindning och är därför tvetydig. • En metod anropas på ett objekt som inte är kompatibelt. Ett TypeError-undantag genereras till exempel om en metod i RegExp-klassen kopplas till ett generiskt objekt och sedan anropas.
URIError	Ett URIError-undantag uppstår när en av de globala URI-hanteringsfunktionerna används på ett sätt som inte är kompatibelt med dess definition.	Ett URIError-undantag kan du få vid följande tillfällen: Ett ogiltigt URI anges för en API-funktion för Flash Player när ett giltigt URI förväntas, t.ex. i `Socket.connect()`.
VerifyError	Ett VerifyError-undantag genereras när en SWF-fil som har ett felaktigt format eller är skadad påträffas.	När en SWF-fil läser in en annan SWF-fil, kan den överordnade SWF-filen fånga upp ett VerifyError-undantag som genererades av den inlästa SWF-filen.

Felklasser i flash.error-paket

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I flash.error-paketet finns felklasser som betraktas som en del av API:t för Flash-miljön. Till skillnad från felklasserna ovan kommunicerar flash.error-paketet felhändelser som är specifika för Flash-miljöer (som Flash Player och Adobe AIR).

Klassnamn	Beskrivning	Anteckningar
EOFError	Ett EOFError-undantag ges om du försöker läsa förbi slutet på tillgängliga data.	T.ex. orsakas ett EOFError när en av läsmetoderna i IDataInput-gränssnittet anropas och det inte finns tillräckligt med data för att tillgodose läsbegäran.
IllegalOperationError	Ett IllegalOperationError-undantag ges om en metod inte implementeras eller om implementeringen inte täcker den aktuella användningen.	Exempel på ogiltiga åtgärder för felundantag kan vara: • En basklass, t.ex. DisplayObjectContainer, ger mer funktionalitet än scenen kan stödja. Om du till exempel försöker hämta eller ställa in en mask på scenen (med `stage.mask`) genererar Flash-miljön ett IllegalOperationError-undantag med meddelandet "The Stage class does not implement this property or method". • En underklass ärver en metod som inte kräver eller erfordrar stöd. • Vissa åtkomstmetoder anropas när Flash Player kompileras utan åtkomststöd. • Funktioner som endast är för redigering anropas från en körningsversion av Flash Player. • Du försöker ange namnet för ett objekt som är placerat på tidslinjen.
IOError	Ett IOError-undantag ges när vissa typer av I/O-undantag inträffar.	Du får detta fel när exempelvis en läs/skriv-åtgärd görs på en socket som inte är ansluten eller vars anslutning har avbrutits.
MemoryError	Ett MemoryError-undantag ges när en begäran om minnestilldelning misslyckas.	ActionScript Virtual Machine 2 har som standard inga gränser för hur mycket minne ett ActionScript-program kan tilldelas. På en stationär dator är fel på grund av minnestilldelningen ovanliga. Ett fel genereras när systemet inte kan tilldela den minneskapacitet som krävs för en åtgärd. På en stationär dator är alltså dessa undantag ovanliga om inte kravet på minneskapacitet är ovanligt stort;. Till exempel är 3 miljarder byte inte möjligt eftersom ett 32-bitars Microsoft® Windows®-program endast har åtkomst till 2 GB för adressutrymmet.
ScriptTimeoutError	Ett ScriptTimeoutError-undantag ges när ett skripttidsgränsintervall på 15 sekunder är uppnått. Genom att använda ett ScriptTimeoutError-undantag kan du hantera skripttidsgränsen på ett bättre sätt. Om det inte finns någon undantagshanterare, öppnar hanteraren för det ej infångade undantaget en dialogruta med ett felmeddelande.	För att förhindra att en oärlig utvecklare fångar upp undantaget och lägger det i en oändlig slinga är det bara det första ScriptTimeoutError-undantaget i ett skript som kan fångas upp. Ett efterföljande ScriptTimeoutError-undantag kan inte fångas upp av koden och går omedelbart vidare till hanteraren för ej infångade undantag.
StackOverflowError	Ett StackOverflowError-undantag ges när stacken som är tillgänglig för skriptet är slut.	Ett StackOverflowError-undantag kan indikera att en oändlig rekursion har inträffat.

Exempel på att hantera fel: programmet CustomErrors

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I programmet CustomErrors visas tekniker som kan användas för egen felhantering när du bygger upp ett program. Detta omfattar följande:

- Validera XML-paket
- Skriva egna fel
- Orsaka egna fel
- Meddela användare när ett fel inträffar

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Du hittar programfiler för CustomErrors i mappen Samples/CustomError. Programmet består av följande filer:

Fil	Beskrivning
CustomErrors.mxml eller CustomErrors.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML)
com/example/programmingas3/errors/ApplicationError.as	En klass som fungerar som huvudfelklass för både klassen FatalError och klassen WarningError.
com/example/programmingas3/errors/FatalError.as	En klass som definierar ett FatalError-fel som genererats av programmet. Klassen utökar den egna klassen ApplicationError.
com/example/programmingas3/errors/Validator.as	En klass som definierar en enskild metod som validerar ett användarangivet XML-paket för employee.
com/example/programmingas3/errors/WarningError.as	En klass som definierar ett WarningError-fel som genererats av programmet. Klassen utökar den egna klassen ApplicationError.

Översikt av programmet CustomErrors

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När programmet läses in anropas metoden `initApp()` för Flex-program eller så körs tidslinjekoden (icke-funktion) för Flash Professional-program. Den här koden definierar XML-paketet som ska verifieras av klassen Validator. Följande kod körs:

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

XML-paketet visas senare i en TextArea-komponentinstans på scenen. Detta gör att du kan ändra XML-paketet innan du försöker omvalidera det.

När användaren klickar på knappen Validate anropas metoden `validateData()`. Den här metoden validerar XML-paket för employee med hjälp av metoden `validateEmployeeXML()` i klassen Validator. I följande exempel visas metoden `validateData()`:

```
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);
    }
}
```

Till att börja med skapas ett temporärt XML-objekt med hjälp av innehållet i TextArea-komponentinstansen `xmlText`. Därefter anropas metoden `validateEmployeeXML()` i den anpassade Validator-klassen (com.example.programmingas3/errors/Validator.as) och det temporära XML-objektet skickas som en parameter. Om XML-paketet är giltigt visas i Label-komponentinstansen `status` ett meddelande om att det lyckades och programmet avslutas. Om metoden `validateEmployeeXML()` orsakar ett eget fel (d.v.s. att ett FatalError, WarningError eller allmänt Error inträffar), körs rätt `catch`-sats och någon av metoderna `showFatalError()`, `showWarningError()` eller `showGenericError()` anropas. Dessa metoder visar ett meddelande i ett textområde med namnet `statusTex` för att meddela användaren att ett speciellt fel har inträffat. Metoderna uppdaterar dessutom Label-komponentinstansen `status` med ett speciellt meddelande.

Om ett allvarligt fel inträffar under ett försök att validera XML-paketet för employee, visas felmeddelandet i textområdet `statusText`, och TextArea-komponentinstansen `xmlText` och Button-komponentinstansen `validateBtn` visas, vilket framgår av exemplet nedan:

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

Om det i stället för ett allvarligt fel inträffar en varning, visas felmeddelandet i TextArea-instansen `statusText`, men varken TextField- eller Button-komponentinstansen `xmlText` visas. Metoden `showWarningError()` visar det anpassade felmeddelandet i textområdet `statusText`. I meddelandet kan dessutom användaren bestämma om han/hon vill fortsätta valideringen av XML-skriptet eller avbryta det. I följande utdrag visas metoden `showWarningError()`:

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

När användaren klickar på knappen Yes eller No anropas metoden `closeHandler()`. I följande utdrag visas metoden `closeHandler()`:

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

Om användaren väljer att avbryta skriptet genom att klicka på Yes uppstår ett allvarligt fel som leder till att programmet stängs ner.

Bygga egna validerare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Validator innehåller metoden `validateEmployeeXML()`. Metoden `validateEmployeeXML()` tar ett enda argument, `employee`, vilket är det XML-paket som du vill validera. Metoden `validateEmployeeXML()` har följande utseende:

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    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);
    }
}
```

För att kunna valideras måste en ”employee” tillhöra ett (och endast ett) ”cost center”. Om ”employee” inte tillhör ett ”cost center” resulterar metoden i ett allvarligt fel, som bubblar upp till metoden `validateData()` i programmets huvudfil. Om "employee" tillhör flera "cost center" uppkommer ett varningsfel. Den sista kontrollen i XML-validatorn är att användaren har endast ett socialförsäkringsnummer definierat (noden `ssn` i XML-paketet). Om det inte finns exakt en `ssn`-nod uppstår ett allvarligt fel.

Du kan lägga till fler kontroller för metoden `validateEmployeeXML()`, om du till exempel vill kontrollera att `ssn`-noden innehåller ett giltigt nummer eller att medarbetaren har angett minst ett telefonnummer och en e-postadress och att båda värdena är giltiga. Du kan också ändra XML-koden så att varje ”employee” har ett unikt ID och ange ett ID för deras chef.

Definiera klassen ApplicationError

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen ApplicationError fungerar som huvudklass för både klassen FatalError och klassen WarningError. Klassen ApplicationError utökar klassen Error och definierar egna metoder och egenskaper, inklusive ett fel-ID, hur allvarligt felet är och ett XML-objekt som innehåller anpassade felkoder och meddelanden. Den här klassen definierar också två statiska konstanter som används för att definiera hur allvarlig varje feltyp är.

Konstruktormetoden för klassen ApplicationError har följande utseende:

```
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>;
}
```

Varje felnod i XML-objektet innehåller en unik nummerkod och ett felmeddelande. Felmeddelandet kan slås upp med hjälp av felkoden och E4X, vilket framgår i följande `getMessageText()`-metod:

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

Metoden `getMessageText()` tar ett enskilt heltalsargument, `id`, och returnerar en sträng. Argumentet `id` är en felkod för felet som ska slås upp. Om `id` 9001 skickas hämtas felet som anger att ”employee” endast ska vara tilldelad till ett ”cost center”. Om flera fel har samma felkod, returnerar ActionScript endast det första felmeddelandet (`message[0]` i det returnerade XMLList-objektet).

Nästa metod i den här klassen, `getTitle()`, tar inga parametrar och returnerar ett strängvärde som innehåller ID:t för det aktuella felet. Detta värde används för att hjälpa dig att identifiera det exakta felet som inträffade under valideringen av XML-paketet. I följande utdrag visas metoden `getTitle()`:

```
public function getTitle():String
{
    return "Error #" + id;
}
```

Den sista metoden i klassen ApplicationError är `toString()`. Den här metoden åsidosätter den funktion som definieras i klassen Error så att du kan anpassa presentationen av felmeddelandet. Metoden returnerar en sträng som identifierar det specifika felnumret och meddelandet.

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

Definiera klassen FatalError

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen FatalError utökar den egna klassen ApplicationError och definierar tre metoder: FatalError-konstruktorn, `getTitle()` och `toString()`. Den första metoden, FatalError-konstruktorn, tar ett heltalsargument, `errorID`, och ställer in hur allvarligt felet är med hjälp av statiska konstantvärden som definieras i klassen ApplicationError, och erhåller det specifika felets felmeddelande genom att anropa metoden `getMessageText()` i klassen ApplicationError. FatalError-konstruktorn har följande utseende:

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

Nästa metod i klassen FatalError, `getTitle()`, åsidosätter metoden `getTitle()`, som definierades tidigare i klassen ApplicationError, och lägger till texten ”-- FATAL” i titeln för att informera användaren om att ett allvarligt fel har inträffat. Metoden `getTitle()` har följande utseende:

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

Den sista metoden i den här klassen, `toString()`, åsidosätter metoden `toString()` som definierades i klassen ApplicationError. Metoden `toString()` har följande utseende:

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

Definiera klassen WarningError

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen WarningError utökar klassen ApplicationError och är näst intill identisk med klassen FatalError, med undantag av några strängförändringar och att den ställer in felets svårighetsgrad till ApplicationError.WARNING i stället för ApplicationError.FATAL, vilket framgår av följande kod:

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

Kapitel 5: Använda reguljära uttryck

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ett reguljärt uttryck beskrivs ett mönster som används för att hitta och arbeta med matchande text i strängar. Reguljära uttryck påminner om strängar, men de kan innehålla specialkoder för att beskriva mönster och repetitioner. Följande reguljära uttryck matchar en sträng som börjar med tecknet A följt av en eller flera siffror:

```
/A\d+/
```

Följande information beskriver den grundläggande syntaxen för att skapa reguljära uttryck. Emellertid kan reguljära uttryck innehålla många svårigheter och nyanser. Du hittar mer utförligt material om reguljära uttryck på webben och i bokhandeln. Tänk på att reguljära uttryck implementeras på olika sätt i olika utvecklingsmiljöer. I ActionScript 3.0 implementeras reguljära uttryck enligt definitionen i språkspecifikationen ECMAScript, version 3 (ECMA-262).

Fler hjälpavsnitt

RegExp

Grunderna för reguljära uttryck

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett reguljärt uttryck beskriver ett teckenmönster. Reguljära uttryck används vanligtvis för att bekräfta att ett textvärde överensstämmer med ett visst mönster (t.ex. att ett angivet telefonnummer har rätt antal siffror) eller för att ersätta delar av ett textvärde som matchar ett visst mönster.

Reguljära uttryck kan vara enkla. Anta att du vill bekräfta att en speciell sträng matchar ”ABC” eller att du vill ersätta varje instans av ”ABC” i en sträng med någon annan text. Du ska då använda följande reguljära uttryck, som definierar mönstret där bokstäverna A, B och C står efter varandra.

```
/ABC/
```

Observera att det reguljära uttryckets litteral urskiljs med ett snedstreck (`/`).

Mönster för reguljära uttryck kan även vara mer svårtydda och ibland ha ett kryptiskt utseende, vilket visas i följande uttryck som används för att matcha en e-postadress:

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

Du kommer vanligtvis att använda reguljära uttryck för att söka efter mönster i strängar och för att ersätta tecken. I dessa fall kommer du att skapa ett objekt för det reguljära uttrycket och använda det som en parameter för en eller flera metoder i klassen String. I följande metod för klassen String används reguljära uttryck som parametrar: `match()`, `replace()`, `search()` och `split()`. Mer information om dessa metoder finns i ”Returnera mönster i strängar och ersätta delsträngar” på sidan 16.

Klassen RegExp innehåller följande metoder: `test()` och `exec()`. Mer information finns i ”Metoder för att använda reguljära uttryck med strängar” på sidan 89.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer om den här funktionen:

Escape-tecken Ett tecken som visar att tecknet som följer ska hanteras som ett metatecken och inte som en litteral. I syntaxen för reguljära uttryck används ett omvänt snedstreck (\) som ett escape-tecken. Det innebär att ett omvänt snedstreck följt av ett annat tecken representerar en specialkod och inte själva tecknet.

Flagga Ett tecken som anger alternativ för hur mönstret för reguljära uttryck ska användas, t.ex. om skillnad ska göras mellan versaler och gemener.

Metatecken Ett tecken som har en speciell betydelse i ett mönster för reguljära uttryck, i stället för att representera den egentliga betydelsen av tecknet i mönstret.

Kvantifierare Ett eller flera tecken som visar hur många gånger en del av mönstret ska upprepas. En kvantifierare kan exempelvis användas för att ange att postnummer i USA ska innehålla fem eller nio siffror.

Reguljärt uttryck En programsats som definierar ett mönster med tecken som kan användas för att bekräfta om andra strängar matchar mönstret eller om delar av strängen ska ersättas.

Syntaxen för reguljära uttryck

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I detta avsnitt beskrivs alla ActionScript-element för syntaxen i reguljära uttryck. Som du kommer att märka kan reguljära uttryck innehålla många svårigheter och nyanser. Du hittar mer utförligt material om reguljära uttryck på webben och i bokhandeln. Tänk på att reguljära uttryck implementeras på olika sätt i olika utvecklingsmiljöer. I ActionScript 3.0 implementeras reguljära uttryck enligt definitionen i språkspecifikationen ECMAScript, version 3 (ECMA-262).

Vanligtvis använder du reguljära uttryck för att matcha mer komplicerade mönster än en enkel teckensträng. I följande reguljära uttryck definieras mönstret, som består av bokstäverna A, B och C efter varandra följt av en siffra:

```
/ABC\d/
```

Koden `\d` representerar ”alla siffror (digit)”. Det omvända snedstrecket (`\`) kallas ”escape-tecken” och tillsammans med tecknet som kommer därefter (i detta fall ”d”) får det en speciell innebörd i reguljära uttryck.

I följande reguljära uttryck definieras mönstret för bokstäverna ABC följt av en siffra (observera asterisken):

```
/ABC\d*/
```

Asterisken (`*`) är ett *metatecken*. Metatecken är tecken som har en speciell innebörd i reguljära uttryck. Asterisken är en speciell typ av metatecken. Den kallas *kvantifierare* och används för att ange antalet repetitioner för ett tecken eller en grupp med tecken. Mer information finns i ”Kvantifierare” på sidan 81.

Förutom dess mönster kan reguljära uttryck innehålla flaggor, som anger hur reguljära uttryck ska matchas. I följande reguljära uttryck används flaggan `i` för att ange att det reguljära uttrycket ska ignorera skiftlägeskänslighet i matchande strängar:

```
/ABC\d*/i
```

Mer information finns i ”Flaggor och egenskaper” på sidan 86.

Du kan använda reguljära uttryck med följande metoder i klassen String: `match()`, `replace()` och `search()`. Mer information om dessa metoder finns i ”Returnera mönster i strängar och ersätta delsträngar” på sidan 16.

Skapa en instans av reguljära uttryck

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns två sätt att skapa instanser för reguljära uttryck. I det ena sättet används snedstreck (`/`) för att avgränsa det reguljära uttrycket; i det andra används konstruktorn `new`. Följande reguljära uttryck är till exempel likvärdiga:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

Snedstrecket urskiljer en litteral för ett reguljärt uttryck på samma sätt som citattecken urskiljer en stränglitteral. Den del av det reguljära uttrycket som finns inom snedstrecken definierar *mönstret*. Reguljära uttryck kan också innehålla flaggor efter det avslutande snedstrecket. Dessa flaggor anses ingå som en del av det reguljära uttrycket, men de ingår inte i mönstret.

När du använder konstruktorn `new` använder du två strängar för att definiera det reguljära uttrycket. Den första strängen definierar mönstret och den andra strängen flaggorna, vilket framgår av följande exempel:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

När det finns ett snedstreck *within* ett reguljärt uttryck, som definierats med hjälp av snedstrecksavgränsare, måste du före snedstrecket skriva ett omvänt snedstreck (`\`) som ett escape-tecken. Följande reguljära uttryck matchar mönstret `1/2`:

```
var pattern:RegExp = /1\/2/;
```

Om du vill använda citattecken *inom* ett reguljärt uttryck, som definierats med konstruktorn `new`, måste du skriva ett omvänt snedstreck (`\`) som ett escape-tecken före citattecknen (precis som för String-litteraler). Följande reguljära uttryck matchar mönstret `eat at "joe's"`:

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

Du ska inte använda omvända snedstreck som escape-tecken tillsammans med citattecken i reguljära uttryck som definieras med snedstreck som avgränsare. Du ska inte heller använda escape-tecken med snedstreck i reguljära uttryck som är definierade med konstruktorn `new`. Följande reguljära uttryck är likvärdiga och de definierar båda mönstret `1/2 "joe's"`:

```
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"', '');
```

Om en metasekvens som börjar med ett omvänt snedstreck (`\`) används i ett reguljärt uttryck som är definierat med konstruktorn `new`, t.ex. `\d` (som matchar alla siffror), skriver du det omvända snedstrecket två gånger som i följande exempel:

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

I detta fall måste du skriva det omvända snedstrecket två gånger eftersom den första parametern till konstruktormetoden `RegExp()` är en sträng och i en stränglitteral måste du skriva det omvända snedstrecket två gånger för att det ska tolkas som ett enkelt omvänt snedstreck.

I följande avsnitt beskrivs syntaxen för att definiera mönster för reguljära uttryck.

Mer information om flaggor finns i ”Flaggor och egenskaper” på sidan 86.

Tecken, metatecken och metasekvenser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det enklaste reguljära uttrycket är det som matchar en sekvens med tecken som i följande exempel.

```
var pattern:RegExp = /hello/;
```

Följande tecken, metatecken, har emellertid en speciell innebörd i reguljära uttryck:

```
^ $ \ . * + ? ( ) [ ] { } |
```

I nästa exempel matchar det reguljära uttrycket bokstaven A följt av inga eller fler instanser av bokstaven B (asterisken är ett metatecken som indikerar repetition), följt av bokstaven C:

```
/AB*C/
```

Om du vill inkludera ett metatecken utan att det ska ha någon speciell innebörd i mönstret för ett reguljärt uttryck måste du använda det omvända snedstrecket (`\`) som escape-tecken. Följande reguljära uttryck matchar bokstaven A, följt av bokstaven B, följt av en asterisk som i sin tur följs av bokstaven C:

```
var pattern:RegExp = /AB\*C/;
```

En *metasekvens*, som ett metatecken, har en speciell innebörd i ett reguljärt uttryck. En metasekvens består av flera tecken. I följande avsnitt beskrivs hur du använder metatecken och metasekvenser.

Om metatecken

I tabellen nedan finns en sammanställning av de metatecken som du kan använda i reguljära uttryck:

Metatecken	Beskrivning
^ (cirkumflex)	Används för att matcha början på strängen. Med taggen `m` (`multiline`) inställd används cirkumflexet även för att matcha början på en rad (se "Flaggor och egenskaper" på sidan 86). Observera, att när cirkumflexet används i början av en teckenklass visas en negation – inte början på en sträng. Mer information finns i "Teckenklasser" på sidan 80.
$(dollartecken)	Används för att matcha slutet på strängen. Med taggen `m` (`multiline`) inställd används `dollartecknet` även för att matcha positionen före ett radmatningstecken (`\n`). Mer information finns i "Flaggor och egenskaper" på sidan 86.
\ (omvänt snedstreck)	Undertrycker den speciella metateckensbetydelsen för specialtecken. Du kan också använda det omvända snedstrecket om du vill använda ett vanligt snedstreck i en litteral för reguljära uttryck som i `/1\/2/` (för att matcha tecknet 1, följt av snedstrecket och tecknet 2).
. (punkt)	Matchar valfritt tecken. En punkt matchar ett radmatningstecken (`\n`) bara om flaggan `s` (`dotall`) är inställd. Mer information finns i "Flaggor och egenskaper" på sidan 86.
* (stjärna)	Matchar det föregående objektet repeterat noll gånger eller flera. Mer information finns i "Kvantifierare" på sidan 81.
+ (plus)	Matchar det föregående objektet repeterat en eller flera gånger, Mer information finns i "Kvantifierare" på sidan 81.
? (frågetecken)	Matchar det föregående objektet repeterat noll eller en gång. Mer information finns i "Kvantifierare" på sidan 81.

Metatecken	Beskrivning
`(` och `)`	Definierar grupper inom det reguljära uttrycket. Du använder grupper för följande: • För att begränsa omfattningen för vertikalstrecket |: `/(a|b|c)d/` • För att definiera omfattningen för en kvantifierare: `/(walla.){1,2}/` • I bakåtreferenser. `\1` i följande reguljära uttryck matchar det som matchades med mönstret i den första parentetiska gruppen: • `/(\w*) is repeated: \1/` Mer information finns i "Grupper" på sidan 83.
`[` och `]`	Definiera en teckenklass som definierar möjliga matchningar för ett enskilt tecken: `/[aeiou]/` matchar vart och ett av de angivna tecknen. Inom teckenklasser använder du ett bindestreck (`-`) för att ange ett teckenintervall: `/[A-Z0-9]/` matchar alla versaler från A till Z eller siffror från 0 till 9. Inom teckenklasser använder du ett omvänt snedstreck för att undertrycka tecknen `]` och `-` tecken: `/[+\-]\d+/` matchar antingen `+` eller `-` före en eller flera siffror. Inom teckenklasserna hanteras andra tecken som vanligtvis är metatecken som vanliga tecken (inte metatecken) utan att något omvänt snedstreck behöver användas. `/[$]/£` matchar antingen `$` eller £. Mer information finns i "Teckenklasser" på sidan 80.
| *(vertikalstreck)*	Används för att kunna matcha uttryck på antingen den vänstra eller högra sidan om tecknet. `/abc|xyz/` matchar antingen `abc` eller `xyz`.

Om metasekvenser

Metasekvenser är tecken som har en speciell innebörd i ett mönster för reguljära uttryck. I tabellen nedan beskrivs dessa metasekvenser:

Metasekvens	Beskrivning

`{n}`
`{n,}`
och
`{n,n}`
	Anger en numerisk kvantifierare eller ett kvantifieringsintervall för föregående objekt:
	`/A{27}/` matchar tecknet `A` repeterat `27` gånger.
	`/A{3,}/` matchar tecknet `A` repeterat `3` eller flera gånger.
	`/A{3,5}/` matchar tecknet `A` repeterat `3` till `5` gånger.
	Mer information finns i "Kvantifierare" på sidan 81.

`\b`	Matchar positionen mellan ett ordtecken och ett icke-ordtecken. Om det första eller sista tecknet i strängen är i ett ordtecken, matchas även början eller slutet av strängen.
`\B`	Matchar positionen mellan två ordtecken. Matchar även positionen mellan två icke-ordtecken.
`\d`	Matchar en decimal siffra.
`\D`	Matchar alla tecken förutom siffror.
`\f`	Matchar arkmatningstecknet.
`\n`	Matchar radmatningstecknet.
`\r`	Matchar vagnretursteckenet.

Metasekvens	Beskrivning
`\s`	Matchar alla tomrum (blanksteg, tabb, ny rad eller returtecknet).
`\S`	Matchar alla tecken förutom tomrum.
`\t`	Matchar tabbtecknet.
`\u`***nnnn***	Matchar Unicode-tecknet med teckenkoden som uttrycks hexadecimalt med *nnnn*. Exempelvis `\u263a` motsvarar "smiley"-tecknet.
`\v`	Matchar ett lodrätt arkmatningstecken.
`\w`	Matchar ett ordtecken (`AZ`–, `az`–, `0-9` eller _). Tänk på att `\w` inte matchar icke-engelska tecken, t.ex. `é`, `ñ` eller `ç`.
`\B`	Matchar alla tecken förutom ordtecken.
`\\x`***nn***	Matchar tecken med det angivna ASCII-värdet såsom det uttrycks hexadecimalt med *nn*.

Teckenklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder teckenklasser för att ange en teckenlista som ska matcha en position i det reguljära uttrycket. Du definierar teckenklasser med hakparenteser (`[` och `]`). I följande reguljära uttryck definieras en teckenklass som matchar `bag`, `beg`, `big`, `bog` eller `bug`:

```
/b[aeiou]g/
```

Escape-sekvenser i teckenklasser

De flesta metatecken och metasekvenser som vanligtvis har en speciell betydelse i ett reguljärt uttryck har *inte* samma betydelse i en teckenklass. I ett reguljärt uttryck används exempelvis asterisken för att repetitioner, men detta gäller inte när asterisken används i en teckenklass. Följande teckenklass matchar asterisken litteralt, men även övriga listade tecken:

```
/[abc*123]/
```

De tre tecknen i tabellen nedan fungerar emellertid som metatecken med en speciell betydelse i teckenklasser:

Metatecken	Betydelse i teckenklasser
`]`	Definierar slutet på teckenklassen.
`-`	Definierar ett teckenintervall (se avsnittet "Teckenintervall i teckenklasser").
`\`	Definierar metasekvenser och upphäver den speciella betydelsen för metatecknen.

För att dessa tecken ska identifieras som litteraler (utan den speciella metateckenbetydelsen) måste du före tecknen skriva ett omvänt snedstreck som escape-tecken. I följande reguljära uttryck finns en teckenklass som matchar dessa fyra symboler(`$`, `\`, `]` eller `-`):

```
/[$\\\]\-]/
```

Förutom metatecknen som behåller specialbetydelsen fungerar följande metasekvenser som metasekvenser inom teckenklasser:

Metasekvens	Betydelse i teckenklasser
`\n`	Matchar ett radmatningstecken.
`\r`	Matchar ett vagnreturstecken.
`\t`	Matchar ett tabbtecken.
`\u`***nnnn***	Matchar tecknet med den angivna Unicode-kodpunkten såsom den uttrycks hexadecimalt med *nnnn*.
`\\x`***nn***	Matchar tecknet med det angivna ASCII-värdet såsom det uttrycks hexadecimalt med *nn*.

Andra reguljära uttryck för metasekvenser och metatecken hanteras i en teckenklass som vanliga tecken.

Teckenintervall i teckenklasser

Använd bindestreck för att ange teckenintervall, t.ex. `A-Z`, `a-z` eller `0-9`. Dessa tecken måste utgöra ett giltigt intervall i teckenuppsättningen. Följande teckenklass matchar alla tecken i intervallet `a-z` och alla siffror:

```
/[a-z0-9]/
```

Du använder ASCII-teckenkoden `\\x`*nn* för att ange ett intervall för ASCII-värden. Följande teckenklass matchar alla tecken från en uppsättning av den utökade ASCII-teckenuppsättningen (t.ex. é och ê):

```
\\x
```

Negerade teckenklasser

När du använder ett cirkumflex (`^`) i början av en teckenklass kommer klassen att negeras, dvs. alla tecken som inte finns med i listan betraktas som en matchning. Följande teckenklass matchar alla tecken *med undantag för* ett gement tecken (`a-z`) eller en siffra:

```
/[^a-z0-9]/
```

Du måste skriva cirkumflexet (`^`) i *början* av en teckenklass för att indikera negationen. Du kan annars lägga till cirkumflexet till tecknen i teckenklassen. Följande teckenklass matchar symbolerna, inklusive cirkumflexet:

```
/[!.,#+*%$&^]/
```

Kvantifierare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder kvantifierare för att ange teckenrepetitioner eller sekvenser i mönster enligt följande:

Metatecken för kvantifierare	Beskrivning
`*` (stjärna)	Matchar det föregående objektet repeterat noll gånger eller flera.
`+` (plus)	Matchar det föregående objektet repeterat en eller flera gånger,
`?` (frågetecken)	Matchar det föregående objektet repeterat noll eller en gång.
{***n***} {***n***,} och {***n***,***n***}	Anger en numerisk kvantifierare eller ett kvantifieringsintervall för föregående objekt: `/A{27}/` matchar tecknet A repeterat 27 gånger. `/A{3,}/` matchar tecknet A repeterat 3 eller flera gånger. `/A{3,5}/` matchar tecknet A repeterat 3 till 5 gånger.

Du kan tillämpa en kvantifierare på ett enskilt tecken, på en teckenklass eller på en grupp.

- `/a+/` matchar tecknet `a` repeterat en eller flera gånger.
- `/\d+/` matchar en eller flera siffror.
- `/[abc]+/` matchar en repetition av en eller flera tecken, där tecknet är antingen `a`, `b` eller `c`.
- `/(very, )*/` matchar order `very` följt av ett komma och ett blanksteg repeterat noll eller flera gånger.

Du kan använda kvantifierare inom parentetiska grupper med kvantifierare. Följande kvantifierare matchar strängar som `word` och `word-word-word`:

```
/\w+(-\w+)*/
```

I reguljära uttryck utförs som standard något som kallas *girig matchning*. När du använder delmönster i det reguljära uttrycket (t.ex. `.*`) görs försök att matcha så många tecken i strängen som möjligt innan matchningen av nästa del av det reguljära uttrycket görs. Titta på följande reguljära uttryck och sträng:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

Det reguljära uttrycket matchar hela strängen:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Anta att du bara vill matcha en `<p>...</p>`-gruppering. Du kan då göra följande:

```
<p>Paragraph 1</p>
```

Skriv ett frågetecken (`?`) efter en kvantifierare för ändra den till en s.k. *lat kvantifierare*. I följande reguljära uttryck används kvantifieraren `*?` för att matcha `<p>` följt av minsta möjliga antal tecken (”lat”), följt av `</p>`:

```
/<p>.*?<\/p>/
```

Tänk på följande när det gäller kvantifierare:

- Kvantifierarna `{0}` och `{0,0}` utesluter inte ett objekt från matchningen.
- Kombinera inte flera kvantifierare som `/abc+*/`.
- Punkten (.). omfattar inte flera rader om inte flaggan `s` (`dotall`) anges. Detta gäller även om den följs av en `*`-kvantifierare. Titta på följande kod:

```
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));
    // output: <p>Test
    //                  Multiline</p>
```

Mer information finns i ”Flaggor och egenskaper” på sidan 86.

Växlingar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd den lodräta linjen `|` i ett reguljärt uttryck för att välja matchningsalternativ för det reguljära uttrycket. Följande reguljära uttryck matchar vart och ett av orden `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Du använder parenteser för att definiera grupper så att du kan begränsa omfattningen av vertikalstrecket |. Följande reguljära uttryck matchar `cat` följt av `nap` eller `nip`:

```
var pattern:RegExp = /cat(nap|nip)/;
```

Mer information finns i ”Grupper” på sidan 83.

Följande två reguljära uttryck är likvärdiga där vertikalstrecket `|` används i det första och en teckenklass i det andra (definierad med `[` och `]`):

```
/1|3|5|7|9/
/[13579]/
```

Mer information finns i ”Teckenklasser” på sidan 80.

Grupper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med parenteser kan du ange en grupp i ett reguljärt uttryck, vilket visas i följande exempel:

```
/class-(\d*)/
```

En grupp är ett delavsnitt i ett mönster. Du använder grupper för bl.a. följande:

- Tillämpa en kvantifierare på mer än ett tecken.
- Urskilja delmönster för växlingar (med hjälp av tecknet `|`).
- Hämta delsträngsmatchningar (t.ex. med hjälp av `\1` i ett reguljärt uttryck för att matcha en tidigare matchad grupp eller genom att använda `$1` på samma sätt i metoden `replace()` i klassen String).

I följande avsnitt finns mer information om hur du använder dessa grupper.

Använda grupper med kvantifierare

Om du inte använder grupper tillämpas kvantifierare på tecknet eller teckenklassen som föregår dem, vilket framgår av exemplet nedan:

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

Du kan emellertid använda en grupp för att tillämpa en kvantifierare på mer än ett tecken eller en teckenklass.

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as ababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

Mer information om kvantifierare finns i ”Kvantifierare” på sidan 81.

Använda grupper med vertikalstrecket (|)

Med grupper kan du definiera teckengrupper för vilka du vill tillämpa vertikalstrecket (`|`) enligt följande:

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

Använda grupper för att hämta delsträngsmatchningar

När du definierar en vanlig parentetisk grupp i ett mönster kan du referera till den senare i det reguljära uttrycket. Detta kallas en *bakåtreferens* och grupperna kallas *hämtade grupper*. I följande reguljära uttryck matchar sekvensen `\1` delsträngar som matchade den hämtade parentetiska gruppen:

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

Du kan ange upp till 99 bakåtreferenser i ett reguljärt uttryck genom att skriva `\1`, `\2`, ... , `\99`.

På samma sätt kan du i metoden `replace()` i strängen String använda `$1–$99` för att infoga hämtade grupper i ersättningssträngen:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

Om du dessutom använder hämtade grupper returnerar metoden `exec()` i klassen RegExp och metoden `match()` i klassen String delsträngarna som matchar de hämtade grupperna:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

Använda icke-hämtade grupper och framåtblickande grupper

En icke-hämtad grupp är en grupp som endast används för gruppering. Den ”samlas” inte in och den matchar inte numrerade bakåtreferenser. Använd `(?:` och `)` för att definiera icke-hämtade grupper enligt följande:

```
var pattern = /(?:com|org|net);
```

Observera till exempel skillnaden mellan att placera `(com|org)` i en hämtad och icke-hämtad grupp (metoden `exec()` visar hämtade grupper efter en avslutad matchning):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

En speciell typ av icke-hämtade grupper är *framåtblickande grupper,* av vilka det finns två typer: den *positivt framåtblickande gruppen* och den *negativt framåtblickande gruppen.*

Använd `(?=` och `)` för att definiera en positiv framåtblickande grupp där ett delmönster i en grupp måste matcha den avsedda positionen. Emellertid kan den delen av strängen som matchar den positiva framåtblickande gruppen också matcha återstående mönster i det reguljära uttrycket. Eftersom `(?=e)` är en positiv framåtblickande grupp i exemplet nedan, kan tecknet `e` som den matchar matcha en efterföljande del i det reguljära uttrycket – i detta fall den hämtade gruppen, `\w*)`:

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Använd `(?!` och `)` för att definiera en negativ framåtblickande grupp som anger att delmönstret i gruppen *inte* behöver matcha den avsedda positionen. Till exempel:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

Använda namngivna grupper

En namngiven grupp är en grupp i ett reguljärt uttryck som har en namngiven identifierare. Använd `(?P<name>` och `)` för att definiera den namngivna gruppen. I följande reguljära uttryck finns en namngiven grupp med en identifierare med namnet `digits`:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

När du använder metoden `exec()`, läggs en matchande namngiven grupp till som en egenskap för arrayen `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Här är visas ett exempel där två namngivna grupper, med identifierarna `name` och `dom`, används:

```
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
```

Obs! *Namngivna grupper ingår inte i språkspecifikationen ECMAScript. De är extrafunktioner i ActionScript 3.0.*

Flaggor och egenskaper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande tabell visas fem flaggor som du kan använda i reguljära uttryck. Varje flagga kan användas som en egenskap i objekt för reguljära uttryck.

Flagga	Egenskap	Beskrivning
`g`	`global`	Matchar mer än en matchning.
`i`	`ignoreCase`	Versalkänslig matchning Gäller för tecknen `A` – `Z` och `a` – `z`, men inte för utökade tecken som `É` och `é`.
`m`	`flerrads`	När flaggan är angiven matchar `$` och `^` början respektive slutet av en rad.
`s`	`dotall`	När flaggan är angiven matchar `.` (punkt) radmatningstecknet (`\n`).
`x`	`extended`	Gör det möjligt att använda utökade reguljära uttryck. Du kan använda blanktecken i reguljära uttryck som ej kommer att ingå som en del av mönstret. Detta gör att du kan skriva en mer lättläst kod i det reguljära uttrycket.

Observera att dessa egenskaper är skrivskyddade. Du kan ange flaggorna (`g`, `i`, `m`, `s`, `x`) när du ställer in en variabel för det reguljära uttrycket enligt följande exempel:

```
var re:RegExp = /abc/gimsx;
```

Du kan emellertid inte direkt ange de namngivna egenskaperna. Följande kod kommer exempelvis att resultera i ett fel:

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

Om du inte anger flaggorna i deklarationen av det reguljära uttrycket kommer de inte att anges och motsvarande egenskaper kommer då att sättas till `false`.

Det finns ytterligare två egenskaper i ett reguljärt uttryck:

- Med egenskapen `lastIndex` anger du indexpositionen i strängen så att den kan användas vid nästa anrop av metoderna `exec()` eller `test()` i ett reguljärt uttryck.
- Med egenskapen `source` anger du strängen som definierar mönsterdelen av det reguljära uttrycket.

g-flaggan (global)

När du *inte* använder `g`-flaggan (`global`) kommer det reguljära uttrycket endast att utföra en matchning. När `g`-flaggan inte ingår i det reguljära uttrycket kommer metoden `String.match()` endast att returnera en matchande delsträng, vilket visas i nästa exempel:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

När `g`-flaggan är angiven kommer metoden `Sting.match()`, som framgår av nästa exempel, att returnera flera matchningar:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

i-flaggan (ignoreCase)

Enligt standarden är matchningar i reguljära uttryck versalkänsliga. När du väljer att använda `i`-flaggan (`ignoreCase`) kommer versalkänsligheten att ignoreras. I nästa exempel kommer gemenen `s` i det reguljära uttrycket inte att matcha versalen `S` (det första tecknet i strängen):

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

När du emellertid väljer att använda `i`-flaggan kommer det reguljära uttrycket att matcha versalen `S`:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

Flaggan `i` ignorerar versalkänsligheten bara för tecknen `A–Z` och `a–z`, men inte för utökade tecken som `É` och `é`.

m-flaggan (multiline)

Om du väljer att inte ställa in `m`-flaggan (`multiline`), kommer `^` att matcha början av strängen och `$` slutet av strängen. Om `m`-flaggan är angiven matchar dessa tecken början respektive slutet av en rad. Titta på följande exempel där det finns ett radmatningstecken:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Trots att `g`-flaggan (`global`) är angiven i det reguljära uttrycket matchar metoden `match()` endast en delsträng, eftersom det bara finns en matchning för `^`, strängens början. Resultatet blir:

```
Test
```

Här är samma kod, men det här gången används `m`-flaggan:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

Nu visas i resultatet orden i början av de båda raderna.

```
Test,Multiline
```

Observera att det bara är tecknet `\n` som signalerar radslut. Följande tecken gör det inte:

- Returtecknet (`\r`)
- Unicode-tecknet för radavskiljare (`\u2028`)
- Unicode-tecknet för stryckeavskiljare (`\u2029`)

s-flaggan (dotall)

Om `s`-flaggan (`dotall`) är angiven, matchar en punkt (`.`) i mönstret för det reguljära uttrycket inte ett radmatningstecken (`\n`). Detta medför att inte finns några matchningar i följande exempel:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Om däremot `s`-flaggan används kommer punkten att matcha ett radmatningstecken:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

I nästa exempel gäller matchningen hela delsträngen inom `<p>`-taggarna, inklusive radmatningstecknet:

```
<p>Test
Multiline</p>
```

x-flaggan (extended)

Reguljära uttryck kan vara svåra att läsa och detta gäller speciellt när de innehåller metasymboler och metasekvenser. Till exempel:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

När du använder `x`-flaggan (`extended`) i ett reguljärt uttryck kommer blanksteg som du skriver i mönstret att ignoreras. Följande reguljära uttryck är identiskt med det föregående:

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

Om du väljer att använda `x`-flaggan och om du vill matcha ett blanksteg måste du före blanksteget skriva ett omvänt snedstreck. Följande två reguljära uttryck är likvärdiga:

```
/foo bar/
/foo \ bar/x
```

Egenskapen lastIndex

Egenskapen `lastIndex` anger indexpositionen i strängen där nästa sökning ska starta. Den här egenskapen påverkar metoderna `exec()` och `test()` vilka båda anropar ett reguljärt uttryck där `g`-flaggan är satt till `true`. Titta på följande kod:

```
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);
}
```

Egenskapen `lastIndex` är enligt standard 0 (sökningen startar vid början av strängen). Efter varje matchning ändras indexpositionen enligt matchningen. Detta innebär att föregående kod kommer att resultera i följande utmatning:

```
0
5
11
18
25
36
44
```

Om `global`-flaggan är `false`, kommer egenskapen `lastIndex` inte att användas eller anges i metoderna `exec()` och `test()`.

Metoderna `match()`, `replace()` och `search()` i klassen String startar alla sökningarna från början av strängen, oberoende av inställningen i egenskapen `lastIndex` i det reguljära uttrycket som används för att anropa metoden. (Emellertid sätter metoden `match()` `lastIndex` till 0.)

Du kan ställa in egenskapen `lastIndex` om du vill justera startpositionen i strängen för reguljär uttrycksmatchning.

Egenskapen source

Med egenskapen `source` anger du strängen som definierar mönsterdelen av det reguljära uttrycket. Till exempel:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Metoder för att använda reguljära uttryck med strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen RegExp innehåller två metoder: `exec()` och `test()`.

Förutom metoderna `exec()` och `test()` i klassen RegExp, innehåller klassen String följande metoder för att matcha reguljära uttryck i strängar: `match()`, `replace()`, `search()` och `splice()`.

Metoden test()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `test()` i klassen RegExp används för att kontrollera om strängarna innehåller en matchning för det reguljära uttrycket, vilket visas i följande exempel:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

Metoden exec()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `exec()` i klassen RegExp används för att kontrollera om det i den skickade strängen finns en matchning av det reguljära uttrycket och den returnerar en array med följande:

- Den matchande delsträngen
- Delsträngsmatchningar för eventuella parentetiska grupper i det reguljära uttrycket

Arrayen innehåller även egenskapen `index` vilken visar indexpositionen för början av delsträngsmatchningen.

Titta på följande kod:

```
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
```

Använd `exec()` flera gånger om du vill matcha flera delsträngar när `g`-flaggan (`global`) är inställd för det reguljära uttrycket:

```
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);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

Strängmetoder där RegExp-parametrar används

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande metod för klassen String används reguljära uttryck som parametrar: `match()`, `replace()`, `search()` och `split()`. Mer information om dessa metoder finns i ”Returnera mönster i strängar och ersätta delsträngar” på sidan 16.

Exempel på reguljära uttryck: en Wiki-parser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Detta enkla exempel på Wiki-textkonvertering visar några användningsområden för reguljära uttryck. Det visar hur du:

- konverterar textrader som matchar ett käll-Wiki-mönster med rätt HTML-sträng
- använder ett reguljärt uttryck för att konvertera URL-mönster till hyperlänkstaggen `<a>` i HTML
- använder ett reguljärt uttryck för att konvertera dollarsträngar (som `"$9.95"`) till eurosträngar (som `"8.24 €"`).

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Du hittar programfiler för WikiEditor i mappen Samples/WikiEditor. Programmet består av följande filer:

Fil	Beskrivning
WikiEditor.mxml eller WikiEditor.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/regExpExamples/WikiParser.as	En klass som innehåller metoder där reguljära uttryck används för att konvertera Wiki-textmönster till motsvarande HTML-strängar.
com/example/programmingas3/regExpExamples/URLParser.as	En klass som innehåller metoder där reguljära uttryck används för att konvertera URL-strängarna till hyperlänkstaggen `<a>` i HTML.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	En klass som innehåller metoder där reguljära uttryck används för att konvertera dollarsträngar till eurosträngar.

Definiera klassen WikiParser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen WikiParser innehåller metoder för att konvertera Wiki-indata till motsvarande HTML-utdata. Detta är inget stabilt Wiki-konverteringsprogram, men det innehåller några bra exempel på hur reguljära uttryck kan användas för mönstermatchning och strängkonvertering.

Konstruktorfunktionen tillsammans med metoden `setWikiData()` initierar en enkel sträng med Wiki-indata enligt följande:

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

När användaren klickar på knappen Test i exempelprogrammet anropas metoden `parseWikiString()` för WikiParser-objektet. Den här metoden anropar ett antal andra metoder vilka i sin tur sätter samma den resulterade HTML-strängen.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

I de anropade metoderna – `parseBold()`, `parseItalic()`, `linesToParagraphs()` och `parseBullets()` – används metoden `replace()` för strängen för att ersätta matchande mönster, definierade i ett reguljärt uttryck, så att Wiki-texten kan omformas till HTML-formaterad text.

Konvertera mönster för fet och kursiv stil

Med metoden `parseBold()` kan du söka efter ett Wiki-textmönster med fet stil (t.ex. `'''foo'''`) och omforma det till dess HTML-motsvarighet (t.ex. `<b>foo</b>`), enligt följande:

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Observera att delen med `(.?*)` i det reguljära uttrycket matchar alla tecken (`*`) mellan de båda definierade `'''`-mönstren. Med händelsen `?Med kvantifieraren` görs en matchning som inte är ”girig” så att för en sträng som `'''aaa''' bbb '''ccc'''`, kommer den första matchande delsträngen att vara `'''aaa'''` och inte den kompletta strängen (vilken startar och slutar med mönstret `'''`).

Parenteser i reguljära uttryck används för att definiera en hämtad grupp och metoden `replace()` refererar till gruppen med hjälp av koden `$1` i ersättningssträngen. `g`-flaggan (`global`) i det reguljära uttrycket gör att metoden `replace()` ersätter alla matchningar i strängen (inte bara den första).

Metoden `parseItalic()` fungerar som metoden `parseBold()` med undantag för att den kontrollerar två apostrofer (`''`) som avgränsare för kursiv text (inte här):

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

Konvertera mönster för punktlistor

I exemplet nedan används metoden `parseBullet()` för att leta efter punktlistemönstret för Wiki (t.ex. `* foo`) och omformar det till dess HTML-motsvarighet (t.ex. `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

Symbolen `^` i början av det reguljära uttrycket matchar radens början. `m`-flaggan (`multiline`) i det reguljära uttrycket leder till att det reguljära uttrycket matchar symbolen `^` mot början av raden, inte bara början av strängen.

Mönstret `\*` matchar en asterisk (det omvända snedstrecket används för att ange att det är en litteral asterisk och inte en `*`-kvantifierare).

Parenteser i reguljära uttryck används för att definiera en hämtad grupp och metoden `replace()` refererar till gruppen med hjälp av koden `$1` i ersättningssträngen. `g`-flaggan (`global`) i det reguljära uttrycket gör att metoden `replace()` ersätter alla matchningar i strängen (inte bara den första).

Konvertera Wiki-styckemönster

Med metoden `linesToParagraphs()` kan du konvertera varje rad i en Wiki-sträng till HTML-stycketaggen `<p>`. Dessa rader i metoden tar bort tomma rader från Wiki-strängen:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Symbolerna `^` och `$` i det reguljära uttrycket matchar början och slutet av en rad. `m`-flaggan (`multiline`) i det reguljära uttrycket leder till att det reguljära uttrycket matchar symbolen ^ mot början av raden, inte bara början av strängen.

Metoden `replace()` används för att ersätta alla matchande delsträngar (tomma rader) med en tom sträng (`""`). `g`-flaggan (`global`) i det reguljära uttrycket gör att metoden `replace()` ersätter alla matchningar i strängen (inte bara den första).

Konvertera URL-adresser till <a>-taggar i HTML

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När användaren klickar på knappen Test i exempelprogrammet och om användaren markerat kryssrutan `urlToATag`, anropas den statiska metoden `URLParser.urlToATag()` så att URL-strängar från Wiki-strängen konverteras till `<a>`-taggar i HTML.

```
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>");
```

Konstruktorfunktionen `RegExp()` används för att sätta ihop ett reguljärt uttryck (`urlPattern`) av ett antal beståndsdelar. Dessa beståndsdelar är alla strängar som definierar delar i mönstret för det reguljära uttrycket.

Den första delen av mönstret för det reguljära uttrycket, som definieras med strängen `protocol`, definierar ett URL-protokoll: antingen `http://` eller `ftp://`. Med parenteserna definieras en icke-hämtad grupp vilket visas med symbolen `?`. Detta betyder att parenteser endast används för att definiera en grupp för växlingsmönstret `|`. Gruppen kommer inte att matcha bakåtreferenskoder (`$1`, `$2`, `$3`) i ersättningssträngen för metoden `replace()`.

För övriga beståndsdelar av det reguljära uttrycket används hämtningsgrupper (vilket visas med parenteser i mönstret), vilka sedan används i bakåtreferenskoderna (`$1`, `$2`, `$3`) i ersättningssträngen för metoden `replace()`.

Den del av mönstret som definieras med strängen `urlPart` matchar *åtminstone* ett av följande tecken: `a-z`, `0-9`, _ eller `-`. Kvantifieraren `+` anger att minst ett tecken matchar. Med händelsen `\.` anger ett obligatoriskt punkttecken (`.`). Återstoden matchar en annan sträng med åtminstone ett av dessa tecken: `a-z`, `0-9`, _ eller `-`.

Den del av mönstret som definieras med strängen `optionalUrlPart` matchar *noll eller flera* av följande tecken: ett punkttecken (`.`) följt av ett eller flera alfanumeriska tecken (inklusive _ och `-`). Kvantifieraren `*` visar att noll eller flera tecken matchade.

I anropet till metoden `replace()` används det reguljära uttrycket och HTML-ersättningssträngen sätts ihop med bakåtreferenser.

Från metoden `urlToATag()` anropas sedan metoden `emailToATag()` där en liknande teknik används för att ersätta e-postmönster med hyperlänkssträngen `<a>` i HTML. De reguljära uttryck som används för att matcha URL-strängar för HTTP, FTP och e-post i den här exempelfilen är enkelt uppbyggda och är endast avsedda som exempel. Det finns andra mer sofistikerade reguljära uttryck för att matcha sådana URL-strängar på ett bättre sätt.

Konvertera dollarsträngar till eurosträngar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När användaren klickar på knappen Test i exempelprogrammet och har markerat kryssrutan `dollarToEuro`, anropar programmet den statiska metoden `CurrencyConverter.usdToEuro()` för att konvertera dollarsträngar (t.ex. `"$9.95"`) till eurosträngar (t.ex. `"8.24 €"`), enligt följande:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

På den första raden definieras ett enkelt mönster för att matcha dollarsträngar. Observera att `$`-tecknet föregås av ett omvänt snedstreck (`\`) som escape-tecken.

I metoden `replace()` används det reguljära uttrycket för mönsterpassning och det anropar funktionen `usdStrToEuroStr()` för att bestämma ersättningssträngen (ett värde i euro).

När ett funktionsnamn används som den andra parametern för metoden `replace()` skickas följande som parametrar till den anropade funktionen:

- Den matchande delen av strängen.
- Eventuellt hämtade parentetiska gruppmatchningar. Hur många argument som skickas på det här sättet beror på antalet hämtade parentetiska gruppmatchningar. Du kan bestämma antalet hämtade parentetiska gruppmatchningar genom att kontrollera `arguments.length -3` i funktionskoden.
- Indexpositionen i strängen där matchningen börjar.
- Den fullständiga strängen.

Med metoden `usdStrToEuroStr()` konverteras mönster för dollarsträngar till eurosträngar enligt följande:

```
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;
}
```

Observera att `args[1]` representerar den hämtade parentetiska gruppen som matchas av det reguljära uttrycket `usdPrice`. Detta är den numeriska delen av dollarsträngen: dvs. dollarbeloppet utan tecknet `$`. I metoden tillämpas en växelkurs och den resulterande strängen returneras (med den efterföljande €-symbolen i stället för den inledande $-symbolen).

Kapitel 6: Arbeta med XML

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 ingår en grupp med klasser baserade på specifikationen ECMAScript för XML (E4X) (ECMA-357 version 2). Dessa klasser innehåller kraftfulla och lättanvända funktioner för att arbeta med XML-data. Om du använder E4X kan du skriva kod med XML-data snabbare än vad som var möjligt med tidigare programmeringstekniker. Ytterligare en fördel är att koden blir lättare att läsa.

Fler hjälpavsnitt

XML, klass

ECMA-357-specifikation

Grunderna i XML

Flash Player 9 och senare, Adobe AIR 1.0 och senare

XML är en standard för att presentera strukturerad information så att den lätt kan hanteras i datorer, dessutom ska koden vara rimligt enkel att skriva och förstå. XML är en förkortning för eXtensible Markup Language. XML-standarden finns tillgänglig på webbplatsen www.w3.org/XML/.

XML ger dig ett standardiserat och bekvämt sätt att kategorisera data så att de blir lättare att läsa, hämta och förändra. I XML används en träd- och taggstruktur som påminner om HTML. Här följer ett enkelt exempel på XML-data:

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML-data kan också vara mer komplext med taggar som kapslas in i andra taggar samt innehålla attribut och andra strukturella komponenter. Här följer ett mer avancerat exempel på XML-data:

```
<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>
```

Observera att detta XML-dokument innehåller andra kompletta XML-strukturer (t.ex. taggen `song` och underordnade taggar). Det visar dessutom andra XML-strukturer som attribut (`tracknumber` och `length` i taggen `song`) och taggar som innehåller andra taggar i stället för data (t.ex. taggen `tracks`).

Komma i gång med XML

För dig som har lite eller ingen erfarenhet av XML följer här en kortfattad beskrivning av de vanligaste aspekterna på XML-data. XML-data skrivs som vanlig oformaterad text med en speciell syntax för att organisera informationen i ett strukturerat format. Vanligtvis kallas en uppsättning med XML-data för ett *XML-dokument*. I XML-format är data organiserade i *element* (vilket är ett enkelt dataobjekt eller en behållare för andra element) med hjälp av en hierarkisk struktur. Alla XML-dokument har ett element som översta nivå eller huvudobjekt. Inom detta rotelement kan det finnas viss information, men det troligaste är att det finns andra element som i sin tur innehåller andra element osv. I XML-dokumentet nedan finns information om ett musikalbum:

```
<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>
```

Varje element urskiljs av en uppsättning *taggar*. Elementens namn omges av vinkelparenteser (mindre-än- och större-än-tecken). Starttaggen visar var elementet startar och innehåller ett elementnamn:

```
<title>
```

Sluttaggen föregås av ett snedstreck:

```
</title>
```

Om ett element inte har något innehåll kan det skrivas som ett tomt element. I XML är elementet:

```
<lastplayed/>
```

identiskt med detta element:

```
<lastplayed></lastplayed>
```

Förutom det som elementen innehåller mellan start- och sluttaggen kan ett element även innehålla andra värden, *attribut*, som definieras i elementets starttagg. Exempelvis definierar nästa XML-element ett attribut med namnet `length`, som innehåller värdet `"4:19"`

```
<song length="4:19"></song>
```

Varje XML-element har ett innehåll vilket antingen är ett värde, ett eller flera XML-element eller ingenting (för ett tomt element).

Lär dig mer om XML

Det finns ett antal böcker och resurser i ämnet om du vill lära dig mer om hur du arbetar med XML. Du kan dessutom titta på följande webbplatser:

- W3Schools XML – självstudier: http://w3schools.com/xml/
- XMLpitstop – självstudier, diskussioner mm: http://xmlpitstop.com/

ActionScript-klasser för att arbeta med XML

ActionScript 3.0 innehåller flera klasser som används för att arbeta med XML-strukturerad information. De två huvudklasserna är:

- XML: Representerar ett enskilt XML-element som kan vara ett XML-dokument med flera underordnade element eller ett element med ett värde i ett dokument.
- XMLList: Representerar en uppsättning av XML-element. Ett XMLList-objekt används när det finns flera XML-element som är relaterade till varandra (på samma nivå och under samma överordnade element i XML-dokumentets hierarki). En XMLList-instans är exempelvis det enklaste sättet att arbeta med den här uppsättningen av XML-element (förmodligen i ett XML-dokument), som i detta exempel:

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

För mer avancerade tillämpningar används XML-namnutrymmen. ActionScript innehåller även klasserna Namespace och QName. Mer information om publicering finns i ”Använda XML-namnutrymmen” på sidan 110.

Förutom de inbyggda klasserna för att arbeta med XML, innehåller ActionScript 3.0 också flera operatorer som tillhandahåller funktioner för att hämta och arbeta med XML-data. Detta sätt att arbeta med XML genom att använda dessa klasser och operatorer kallas ECMAScript för XML (E4X) och det definieras i specifikationen för ECMA-357, version 2.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kan stöta på när du programmerar hanteringsrutiner för XML:

Element Ett enskilt objekt i ett XML-dokument som identifieras av innehållet mellan start- och sluttaggen (inklusive taggarna). XML-element kan innehålla textdata eller andra element, eller vara tomt.

Tomt element Ett XML-element som inte innehåller underordnade element. Tomma element skrivs ofta som självstängande taggar (t.ex. `<element/>`).

Dokument En enskild XML-struktur. Ett XML-dokument kan innehålla ett antal element (eller bestå av ett tomt element), men ett XML-dokument måste ha ett toppnivåelement som innehåller alla andra element i dokumentet.

Nod Ett annat namn för ett XML-dokument.

Attribut Ett namngivet värde associerat med ett element som skrivs in i elementets starttagg med formatet `attributename="value"`, i stället för att skrivas som ett separat underordnat element kapslat i elementet.

E4X-sättet att arbeta med XML

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I specifikationen ECMAScript för XML definieras en uppsättning klasser och funktioner för att arbeta med XML-data. Dessa klasser och funktioner kallas gemensamt för *E4X*. I ActionScript 3.0 finns följande E4X-klasser: XML, XMLList, QName och Namespace.

Metoderna, egenskaperna och operatorerna i klasserna E4X har skapats med tanke på följande:

- Enkelhet – Där det är möjligt gör E4X det enklare att skriva och förstå kod för att arbeta med XML-data.
- Konsekvens – Metoderna och tankegångarna bakom E4X är konsekventa och överensstämmer med andra delar av ActionScript.
- Förtrogenhet – Du använder välkända operatorer när du arbetar med XML-data, t.ex. punktoperatorn (`.`).

Obs! *Det finns en annan XML-klass i ActionScript 2.0. I ActionScript 3.0 har den klassen ändrat namn till XMLDocument så att den inte kommer i konflikt med XML-klassen i ActionScript 3.0 som är en del av E4X. I ActionScript 3.0 ingår de gamla klasserna – XMLDocument, XMLNode, XMLParser och XMLTag – i flash.xml-paketet för att produkten ska vara bakåtkompatibel. De nya E4X-klasserna är huvudklasser så du behöver inte importera ett paket för att använda dem. Information om de äldre XML-klasserna i ActionScript 2.0 finns i flash.xml package i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.*

Här följer ett exempel på hur du använder E4X för att hantera data:

```
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>
```

Det är vanligt att programmet hämtar XML-data från en extern källa, t.ex. en webbtjänst eller en RSS-matning. Men i de kodexempel som anges här tilldelas XML-data som litteraler.

Som framgår av följande exempel innehåller E4X vissa intuitiva operatorer, som punktoperatorn (`.`) och attributidentifieraroperatorn (`@`), för att få åtkomst till egenskaper och attribut i XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

Använd metoden `appendChild()` för att tilldela en ny underordnad nod till XML enligt kodutdraget nedan:

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Använd operatorerna `@` och `.` inte bara för att läsa data utan också för att tilldela data enligt följande exempel:

```
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";
```

Använd en `for`-slinga för att iterera igenom XML-noder som i följande exempel:

```
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-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett XML-objekt kan representera element, attribut, kommentarer, bearbetningsinstruktioner eller textelement.

Ett XML-objekt klassificeras antingen som att det har ett *enkelt innehåll* eller ett *komplext innehåll*. Ett XML-objekt som har underordnade noder klassificeras som att det har ett komplext innehåll. XML-objektet sägs ha ett enkelt innehåll om det är ett av följande: ett attribut, en kommentar, en bearbetningsinstruktion eller en textnod.

I nästa exempel innehåller XML-objektet ett komplext innehåll, inklusive en kommentar och en bearbetningsinstruktion:

```
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>
```

Som framgår av nästa exempel kan du nu använda metoderna `comments()` och `processingInstructions()` för att skapa ett nytt XML-objekt, en kommentar och en bearbetningsinstruktion:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML-egenskaper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen XML har fem statiska egenskaper:

- Egenskaperna `ignoreComments` och `ignoreProcessingInstructions` används för att bestämma om kommentarer eller bearbetningsinstruktioner ska ignoreras när XML-objektet tolkas.
- Egenskapen `ignoreWhitespace` används för att bestämma om blanktecken ska ignoreras i elementtaggar och om inbäddade uttryck endast ska separeras med tomrumstecken.
- Egenskaperna `prettyIndent` och `prettyPrinting` används för att formatera text som returneras av metoderna `toString()` och `toXMLString()` i klassen XML.

Information om dessa egenskaper finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

XML-metoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande metoder använder du för att arbeta med XML-objekt och hierarkiska strukturer:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`

- `parent()`
- `prependChild()`

Följande metoder använder du för att arbeta med objektattribut i XML:

- `attribute()`
- `attributes()`

Följande metoder använder du för att arbeta med objektegenskaper i XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Följande metoder använder du för att arbeta med kvalificerade namn och namnutrymmen:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Följande metoder använder du för att arbeta med och bestämma vissa typer av XML-innehåll:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Följande metoder använder du för att arbeta med strängkonvertering och formatering av XML-objekt:

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Dessutom finns ytterligare ett antal metoder:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Information om dessa metoder finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

XMLList-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En XMLList-instans representerar en godtycklig samling av XML-objekt. Den kan innehålla fullständiga XML-dokument, XML-fragment eller resultatet från en XML-fråga.

Följande metoder använder du för att arbeta med XMLList-objekt och hierarkiska strukturer:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Följande metoder använder du för att arbeta med objektattribut i XMLList:

- `attribute()`
- `attributes()`

Följande metoder använder du för att arbeta med XMLList-egenskaper:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Följande metoder använder du för att arbeta med och bestämma vissa typer av XML-innehåll:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Följande metoder använder du för att arbeta med strängkonvertering och formatering av XMLList-objekt:

- `normalize()`
- `toString()`
- `toXMLString()`

Dessutom finns ytterligare ett antal metoder:

- `contains()`

- `copy()`
- `length()`
- `valueOf()`

Information om dessa metoder finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Om ett XMLList-objekt innehåller exakt ett XML-element kan du använda alla egenskaper och metoder i klassen XML eftersom en XMLList med ett XML-element hanteras på samma sätt som ett XML-objekt. I följande exempel använder du metoden `appendChild()` från klassen XML eftersom `doc.div` är ett XMLList-objekt som innehåller ett element:

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Mer information om XML-egenskaper och -metoder finns i ”XML-objekt” på sidan 99.

Initiera XML-variabler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan tilldela en XML-litteral till ett XML-objekt enligt följande:

```
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>
```

Som framgår av nästa kodutdrag kan du använda konstruktorn `new` för att skapa en instans av ett XML-objekt från en sträng som innehåller XML-data:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Du kommer att få ett körningsfel om de XML-data som finns i strängen inte är väl utformade (t.ex. om en sluttagg saknas).

Du kan också skicka data som referens (från andra variabler) till ett XML-objekt. Detta visas i följande exempel:

```
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())
    // Output: <item id="5">Chicken</item>
```

Om du vill läsa in XML-data från en webbadress använder du klassen URLLoader som i följande exempel:

```
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!");
    }
}
```

Du använder klassen XMLSocket när du vill läsa in XML-data från en socketanslutning. Mer information finns i XMLSocket class i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Samla in och omforma XML-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd metoderna `prependChild()` eller `appendChild()` för att lägga till egenskaper i början av en XML-objektslista med egenskaper, enligt följande exempel:

```
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>
```

Använda metoderna `insertChildBefore()` eller `insertChildAfter()` för att lägga till egenskaper före eller efter en angiven egenskap, enligt följande:

```
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>)
```

Som framgår av nästa exempel kan du också använda klammerparenteser ({ och }) för att skicka data som referens (från andra variabler) när du konstruerar XML-objekt:

```
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)
}
```

Du kan tilldela egenskaper och attribut till ett XML-objekt med hjälp av operatorn `=`, vilket visas i nästa exempel:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Detta ställer in XML-objektet `x` till följande:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Med operatorerna + och += kan du sammanfoga XMLList-objekt:

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Detta ställer in XMLList-objektet `xList` till följande:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Gå igenom XML-strukturer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I XML finns en kraftfull funktion för att skapa komplexa, kapslade data via en linjär sträng med tecken. När du läser in data i ett XML-objekt, tolkas dessa data i ActionScript och läses sedan in i dess hierarkiska struktur i minnet (eller så skickas ett körningsfel om aktuella XML-data inte var väl utformade).

Operatorerna och metoderna i XML- och XMLList-objekten gör det enkelt att gå igenom strukturen med XML-data.

Använd punktoperatorn (.) och dubbelpunktsoperatorn (..) för att få åtkomst till underordnade egenskaper i ett XML-objekt. Se följande XML-objekt:

```
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>
```

Objektet `myXML.book` är ett XMLList-objekt som innehåller underordnade egenskaper till `myXML`-objekt med namnet `book`. Detta är två XML-objekt, som matchar två `book`-egenskaper av `myXML`-objektet.

Objektet `myXML..lastName` är ett XMLList-objekt som innehåller egenskaper för dubbelpunktsoperatorn med namnet `lastName`. Detta är två XML-objekt, som matchar två `lastName` av `myXML`-objektet.

Objektet `myXML.book.editor.lastName` är ett XMLList-objekt som innehåller underordnade objekt med namnet `lastName` som är underordnat ett med namnet `editor` som i sin tur är underordnat ett med namnet `book` i objektet `myXML`: i detta finns ett XMLList-objekt som bara innehåller ett XML-objekt (egenskapen `lastName` med värdet ”`Case`”).

Åtkomst till överordnade och underordnade noder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `parent()` returnerar det överordnade objektet till ett XML-objekt.

Du använder heltalsindexvärdet för en underordnad lista för att få åtkomst till ett specifikt underordnat objekt. Ett XML-objekt av typen `myXML`, kan exempelvis ha två underordnade egenskaper med namnet `book`. Varje underordnad egenskap med namnet `book` har ett associerat indexvärde:

```
myXML.book[0]
myXML.book[1]
```

Om du vill få åtkomst till en indirekt underordnad egenskap anger du ett specifikt indexvärde för namnen både på den underordnade och på den indirekt underordnade egenskapen, enligt följande:

```
myXML.book[0].title[0]
```

Om du emellertid endast har en underordnad till `x.book[0]` med namnet `title`, kan du utelämna indexreferensen enligt följande:

```
myXML.book[0].title
```

Om det på liknande sätt bara finns en underordnad ”book” till objektet `x`, och om detta underordnade objekt endast har ett titelobjekt, kan du utelämna båda indexreferenserna, enligt följande:

```
myXML.book.title
```

Du använder metoden `child()` för att med hjälp av namnen navigera till det underordnade objektet utifrån en variabel eller ett uttryck, vilket visas i nästa exempel:

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Hämta attribut

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Symbolen `@` (attributidentifieraroperatorn) använder du för att hämta attribut i ett XML- eller XMLList-objekt, vilket visas i nästa exempel:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Om du använder jokertecknet `*` tillsammans med symbolen `@` kan du hämta alla attribut i ett XML- eller XMLList-objekt, vilket framgår av följande exempel:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Med metoderna `attribute()` eller `attributes()` kan du hämta ett specifikt attribut eller alla attribut i ett XML- eller XMLList-objekt, vilket visas i exemplet nedan:

```
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
```

Tänk på att du också kan använda följande syntax för att hämta attribut:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Dessa båda är likvärdiga med `employee.@id`. Syntaxen `employee.@id` är däremot att föredra.

Filtrera med attribut eller elementvärden

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda parentesoperatorer – `(` och `)` – för att filtrera element med hjälp av ett specifikt elementnamn eller attributvärde. Se följande XML-objekt:

```
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>
```

Följande uttryck är alla giltiga:

- `x.employee.(lastName == "McGee")` – Detta är den andra `employee`-noden.
- `x.employee.(lastName == "McGee").firstName` – Detta är egenskapen `firstName` för den andra `employee`-noden.
- `x.employee.(lastName == "McGee").@id` – Detta är värdet på attributet `id` för den andra `employee`-noden.
- `x.employee.(@id == 347)` – Den första `employee`-noden.
- `x.employee.(@id == 347).lastName` – Detta är egenskapen `lastName` för den första `employee`-noden.
- `x.employee.(@id > 300)` – Detta är en XMLList med båda `employee`-egenskaperna.
- `x.employee.(position.toString().search("analyst") > -1)` – Detta är en XMLList med båda `position`-egenskaperna.

Om du försöker filtrera attribut eller element som inte finns inträffar ett undantag. På den sista raden i exemplet nedan genererar följande kod ett fel eftersom det inte finns något `id`-attribut i det andra `p`-elementet:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

Detsamma gäller för den sista raden i exemplet nedan där följande kod leder till ett fel eftersom det inte finns någon `b`-egenskap i det andra `p`-elementet.

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Du kan undvika dessa fel genom att använda metoderna `attribute()` och `elements()` för att identifiera matchande egenskaper, enligt följande exempel:

```
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'));
```

Du kan också använda metoden `hasOwnProperty()` som i följande exempel:

```
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'));
```

Använda for..in- och for each..in-satserna

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript 3.0 innehåller programsatserna `for..in` och `for each..in` som du använder för att iterera igenom XMLList-objekt. Ett XML-objekt, `myXML` och XMLList-objektet, `myXML.item`. XMLList-objektet `myXML.item`, består av två `item`-noder av XML-objektet.

```
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>;
```

Slingan `for..in` itererar du igenom en uppsättning egenskapsnamn i en XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

Slingan `for each..in` gör att du kan iterera igenom egenskaperna i XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

Använda XML-namnutrymmen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Namnutrymmen i XML-objekt (eller dokument) används för att identifiera de data som finns i objektet. Om du till exempel skickar och levererar XML-data till en webbtjänst där meddelandeprotokollet SOAP används, ska du deklarera namnutrymmet i starttaggen i XML enligt följande:

```
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>;
```

Namnutrymmet har ett prefix, `soap` och ett URI som definierar namnutrymmet, `http://schemas.xmlsoap.org/soap/envelope/`.

I ActionScript 3.0 finns klassen Namespace som används för att arbeta med XML-namnutrymmen. I XML-objektet i det föregående exemplet använder du klassen Namespace på följande sätt:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: 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";
```

Klassen XML innehåller följande metoder för att arbeta med namnutrymmen: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` och `setNamespace()`.

Med `default xml namespace` kan du tilldela ett standardnamnutrymme för XML-objekt. I nästa exempel har både `x1` och `x2` samma standardnamnutrymme:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML-typkonvertering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det går att konvertera XML- och XMLList-objekt till strängvärden. Du kan också konvertera strängar till XML-objekt och XMLList-objekt. Kom ihåg att alla XML-attributvärden, namn och textvärden är strängar. I kommande avsnitt beskrivs dessa former av XML-typkonvertering.

Konvertera XML- och XMLList-objekt till strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klasserna XML och XMLList innehåller metoderna `toString()` och `toXMLString()`. Metoden `toXMLString()` returnerar en sträng som innehåller alla taggar, attribut, namnutrymmesdeklarationer och innehållet i XML-objektet. För XML-objekt med komplext innehåll (underordnade element) fungerar metoden `toString()` på exakt samma sätt som metoden `toXMLString()`. För XML-objekt med enkelt innehåll (de som endast innehåller ett textelement) returnerar metoden `toString()` endast texten i elementet, vilket visas i nästa exempel:

```
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
```

Om du använder metoden `trace()`, utan att ange `toString()` eller `toXMLString()`, konverteras data vanligtvis med metoden `toString()`, vilket visas nedan:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

Om du använder metoden `trace()` för att felsöka koden bör du använda metoden `toXMLString()` så att metoden `trace()` ger mer kompletta data.

Konvertera strängar till XML-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda konstruktorn `new XML()` om du vill skapa ett XML-objekt från en sträng på följande sätt:

```
var x:XML = new XML("<a>test</a>");
```

Om du försöker konvertera en sträng till XML från en sträng som representerar ogiltig XML eller XML som inte är bra utformad, kommer ett körningsfel i enlighet med nästa exempel att uppstå:

```
var x:XML = new XML("<a>test"); // throws an error
```

Konvertera attributvärden, namn och textvärden från strängar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla XML-attributvärden, namn och textvärden har datatypen String och du kan ibland behöva konvertera dessa till andra datatyper. I följande exempel används funktionen `Number()` för att konvertera textvärden till tal:

```
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); // 4.95;
```

Om funktionen `Number()` inte användes i den här koden skulle +-operatorn tolkas som en strängsammanfogningsoperator och metoden `trace()` i sista raden skulle ge följande resultat:

```
01.003.95
```

Läsa externa XML-dokument

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder klassen URLLoader för att läsa in XML-data från en webbplats. Om du använder följande kod i ditt program ska du ersätta värdet `XML_URL` i exemplet med en giltig webbadress:

```
import flash.events.Event;
import flash.net.URLLoader;

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(Event.COMPLETE, xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

Du använder klassen XMLSocket för att ställa in en asynkron XML-socketanslutning mot en server. Mer information finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Exempel på XML i ActionScript: läsa in RSS-data från Internet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet RSSViewer är ett exempel som visar hur du kan arbeta med XML i ActionScript samt hur du:

- använder XML-metoder för att gå igenom XML-data i form av en RSS-matning
- använder XML-metoder för att samla in XML-data i form av HTML för att sedan användas i textfält.

RSS-formatet är vanligt förekommande för att hämta nyheter via XML. En enkel RSS-datafil kan ha följande utseende:

```
<?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>
```

I programmet SimpleRSS läses RSS-data från Internet in. Dessa data tolkas för rubriken, länkar och beskrivningar, och därefter returneras dessa data. Klassen SimpleRSSUI ger dig gränssnittet och anropar klassen SimpleRSS där all XML-bearbetning sker.

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Du hittar programfiler för RSSViewer i mappen Samples/RSSViewer. Programmet består av följande filer:

Fil	Beskrivning
RSSViewer.mxml eller RSSViewer.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/rssViewer/RSSParser.as	En klass som innehåller metoder som använder E4X för att gå igenom RSS-data (XML) och generera motsvarande HTML-representation.
RSSData/ak.rss	En RSS-exempelfil. Programmet har inställningar för att läsa RSS-data från webben via en Flex RSS-matning som tillhandahålls av Adobe. Du kan emellertid ändra programmet så att det läser RSS-data från detta dokument, som använder ett något annorlunda schema än det som finns i Flex RSS-matningen.

Läsa och tolka XML-data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen RSSParser innehåller en `xmlLoaded()`-metod som konverterar RSS-indata, i variabeln `rssXML`, som sparas i en sträng som innehåller HTML-formaterade utdata, `rssOutput`.

I början av metoden anger koden standard-XML-namnutrymmet om källans RSS-data innehåller ett standardnamnutrymme:

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

Nästa rad skapar sedan en slinga genom innehållet för källans XML-data och varje underordnad egenskap med namnet `item` undersöks:

```
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);
}
```

De första tre raderna anger strängvariablerna så att de representerar titel-, beskrivnings- och länkegenskaperna för egenskapen `item` för XML-datan. Nästa rad anropar metoden `buildItemHTML()` så att HTML-data hämtas i form av ett XMLList-objekt och för detta används tre nya strängvariabler som parametrar.

Samla in XMLList-data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

HTML-informationen (ett XMLList-objekt) har följande format:

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

De första raderna i metoden rensar standard-XML-namnutrymmet:

```
default xml namespace = new Namespace();
```

Direktivet `default xml namespace` omfattar funktionens blocknivå. Det innebär att deklarationens omfattning är metoden `buildItemHTML()`.

Med de rader som följer samlas XMLList in utifrån strängargumenten som skickas till funktionen:

```
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;
```

Detta XMLList-objekt representerar strängdata och är lämpligt för ett HTML-textfält för ActionScript.

Metoden `xmlLoaded()` använder returvärdet för metoden `buildItemHTML()` och konverterar det till en sträng:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Extrahera titeln för RSS-matningen och skicka en anpassad händelse

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `xmlLoaded()` anger en `rssTitle`-strängvariabel baserat på informationen i XML-datan för källan:

```
rssTitle = rssXML.channel.title.toString();
```

Slutligen genererar metoden `xmlLoaded()` en händelse som meddelar programmet att data har tolkats och är tillgängliga:

```
            dataWritten = new Event("dataWritten", true);
```

Kapitel 7: Använda systemspecifik JSON-funktionalitet

I ActionScript 3.0 finns ett systemspecifikt API för kodning och avkodning av ActionScript-objekt med hjälp av JSON-formatet (JavaScript Object Notation). JSON-klassen och de medlemsfunktioner som stöds uppfyller med några få undantag den femte upplagespecifikationen av ECMA-262.

Adobe-experten Todd gör en jämförelse mellan ett systemspecifikt JSON API och JSON-klassen as3corelib från tredje part. Se Working with Native JSON in Flash Player 11.

Fler hjälpavsnitt

JSON

Översikt av JSON-API

ActionScript JSON API består av JSON-klassen och medlemsfunktioner för `toJSON()` i fem systemspecifika klasser. För program som kräver en egen JSON-kodning för någon klass, ger ActionScript-ramverket möjlighet att åsidosätta standardkodningen.

Klassen JSON hanterar internt import och export för alla ActionScript-klasser som inte har en `toJSON()`-medlem. I sådana fall går JSON igenom de publika egenskaperna för varje objekt som upptäcks. Om ett objekt innehåller andra objekt, repeteras JSON för de kapslade objekten och utför samma genomgång. Om något av objekten tillhandahåller en `toJSON()`-metod, kommer den anpassade metoden för JSON att användas i stället för dess interna algoritm.

JSON-gränssnittet består av en kodningsmetod, `stringify()`, och en avkodningsmetod, `parse()`. Dessa metoder ger en parameter som gör att du kan infoga egen logik i arbetsflödet för JSON-kodning/-avkodning. För `stringify()` heter parametern `replacer` och för `parse()` `reviver`. Dessa parametrar tar emot en funktionsdefinition med två argument med hjälp av följande signatur:

```
function(k, v):*
```

toJSON()-metoder

Signaturen för `toJSON()`-metoderna är

```
public function toJSON(k:String):*
```

`JSON.stringify()` anropar `toJSON()`, om den finns, för varje publik egenskap som upptäcks under genomgången av ett objekt. Egenskapen består av ett nyckelvärdepar. När `stringify()` anropar `toJSON()`, skickas nyckeln, `k`, för egenskapen som för närvarande undersöks. I en vanlig `toJSON()`-implementation utvärderas varje egenskapsnamn och returnerar den önskade kodningen av värdet.

Metoden `toJSON()` kan returnera alla typer av värden (anges med *), inte bara en sträng. Den här variabla returtypen tillåter `toJSON()` att returnera ett objekt om det skulle behövas. Om till exempel en egenskap för din klass innehåller ett objekt från ett annat tredjepartsbibliotek, kan du returnera objektet när `toJSON()` upptäcker egenskapen. JSON kommer sedan att repeteras i tredjepartsobjeket. Kodningsprocessen ser ut så här:

- Om `toJSON()` returnerar ett objekt som inte utvärderas som en sträng, repeteras `stringify()` i detta objekt.

- Om `toJSON()` returnerar en sträng för `stringify()` över det värdet till en annan sträng, returnerar den överförda strängen och fortsätter sedan till nästa värde.

I många fall är det bättre att ett objekt returneras än att en JSON-sträng som skapats av ditt program returneras. Returnerandet av ett objekt utnyttjar den inbyggda JSON-kodningsalgoritmen och gör det även möjligt för JSON att gå igenom kapslade objekt rekursivt.

Metoden `toJSON()` är inte definierad i klassen Object och inte heller i de flesta andra inbyggda klasser. Dess frånvaro leder till att JSON utför standardgenomgången av objektets publika egenskaper. Om du vill kan du även använda `toJSON()` för att visa objektets privata egenskaper.

Vissa inbyggda klasser kan medföra problem som ActionScript-biblioteken inte kan lösa effektivt för alla tänkbara användningssätt. För dessa klasser erbjuder ActionScript en enkel implementation som kan återimplementeras på klienter för att passa för de behov som finns. Klasserna som tillhandahåller enkla `toJSON()`-medlemmar är:

- ByteArray
- Date
- Dictionary
- XML

Du kan göra en underklass av ByteArray-klassen för att åsidosätta dess `toJSON()`-metod eller så kan du omdefiniera dess prototyp. Klasserna Date och XML, som deklareras sist, kräver att du använder klassprototypen för att omdefiniera `toJSON()`. Klassen Dictionary deklareras dynamiskt, vilket ger dig extra frihet att åsidosätta `toJSON()`.

Definiera anpassade JSON-beteenden

Om du vill implementera egen JSON-kodning och avkodning för inbyggda klasser kan du välja mellan flera alternativ. Du kan:

- definiera eller åsidosätta `toJSON()` för din anpassade underklass för en icke-slutlig inbyggd klass,
- definiera eller definiera om `toJSON()` för klassens prototyp,
- definiera en `toJSON`-egenskap för en dynamisk klass,
- använda parametrarna `JSON.stringify() replacer` och `JSON.parser() reviver`.

Fler hjälpavsnitt

ECMA-262, femte upplagan

Definiera toJSON() på prototypen för en inbyggd klass

Den systemspecifika JSON-implementeringen i ActionScript speglar ECMAScript JSON-förhållandet definierad i ECMA-262, femte upplagan. Efterson det inte finns stöd för klasser i ECMAScript, definieras JSON-beteendet i ActionScript utifrån prototypbaserade skickningar. Prototyper är föregångare till ActionScript 3.0-klasser, som tillåter simulerade arv, tillägg av medlemmar och omdefinitioner.

ActionScript tillåter att du definierar eller omdefinierar `toJSON()` på prototypen för valfri klass. Denna behörighet gäller även för klasser som är märkta som slutgiltiga. När du definierar `toJSON()` för en klassprototyp, kommer definitionen att gälla för alla instanser av klassen inom programmets räckvidd. Här visas till exempel hur du kan definiera en `toJSON()`-metod för prototypen MovieClip:

```
MovieClip.prototype.toJSON = function(k):* {
    trace("prototype.toJSON() called.");
    return "toJSON";
}
```

När programmet sedan anropar `stringify()` i en MovieClip-instans, returnerar `stringify()` utdata för din `toJSON()`-metod:

```
var mc:MovieClip = new MovieClip();
var js:String = JSON.stringify(mc); //"prototype toJSON() called."
trace("js: " + js); //"js: toJSON"
```

Du kan också åsidosätta `toJSON()` i inbyggda klasser som definierar metoden. I följande kod åsidosätts till exempel `Date.toJSON()`:

```
Date.prototype.toJSON = function (k):* {
    return "any date format you like via toJSON: "+
        "this.time:"+this.time + " this.hours:"+this.hours;
}
var dt:Date = new Date();
trace(JSON.stringify(dt));
// "any date format you like via toJSON: this.time:1317244361947 this.hours:14"
```

Definiera eller åsidosätta toJSON() på klassnivå

Program behöver inte alltid använda prototyper för att omdefiniera `toJSON()`. Du kan också definiera `toJSON()` som en medlem i en underklass, om den överordnade klassen inte har markerats som slutlig. Du kan till exempel utöka klassen ByteArray och definiera en offentlig `toJSON()`-funktion:

```
package {

    import flash.utils.ByteArray;
    public class MyByteArray extends ByteArray
    {
        public function MyByteArray() {
        }

        public function toJSON(s:String):*
        {
            return "MyByteArray";
        }

    }
}


var ba:ByteArray = new ByteArray();
trace(JSON.stringify(ba)); //"ByteArray"
var mba:MyByteArray = new MyByteArray(); //"MyByteArray"
trace(JSON.stringify(mba)); //"MyByteArray"
```

Om en klass är dynamisk kan du lägga till en `toJSON`-egenskap till ett objekt i denna klass och tilldela en funktion till den enligt följande:

```
var d:Dictionary = new Dictionary();
trace(JSON.stringify((d))); // "Dictionary"
d.toJSON = function(){return {c : "toJSON override."};} // overrides existing function
trace(JSON.stringify((d))); // {"c":"toJSON override."}
```

Du kan åsidosätta, definiera eller definiera om `toJSON()` för alla ActionScript-klasser. De flesta inbyggda ActionScript-klasser definierar emellertid inte `toJSON()`. Klassen Object definierar inte `toJSON` i dess standardprototyp och deklarerar den inte som en klassmedlem. Endast en handfull inbyggda klasser definierar metoden som en prototypfunktion. Således kan du i de flesta klasser inte åsidosätta `toJSON()` på traditionellt sätt.

Inbyggda klasser som inte definierar `toJSON()` serialiseras till JSON via den interna JSON-implementeringen. Undvik om möjligt att ersätta denna inbyggda funktion. Om du definierar en `toJSON()`-medlem använder JSON-klassen din logik i stället för dess egna funktioner.

Använda parametern replacer för JSON.stringify()

Att åsidosätta `toJSON()` på en prototyp är användbart när du vill ändra en klass JSON-exportbeteende i hela programmet. I vissa klasser kanske din exportlogik endast gäller speciella fall under kortvariga villkor. För att tillgodose sådana små förändringar kan du använda parametern `replacer` i metoden `JSON.stringify()`.

Metoden `stringify()` tillämpar funktionen som skickas med `replacer`-parametern till objektet som kodas. Signaturen för denna funktion påminner om den som gäller för `toJSON()`:

```
function (k,v):*
```

Till skillnad från `toJSON()`, kräver funktionen `replacer` värdet, `v`, samt nyckeln, `k`. Den här skillnaden är nödvändig eftersom `stringify()` definieras för ett statiskt JSON-objekt i stället för att objektet kodas. När `JSON.stringify()` anropar `replacer(k,v)`, kommer det att gå igenom det ursprungliga indataobjektet . Den implicita `this`-parametern, som skickas till funktionen `replacer`, refererar till objektet som håller nyckeln och värdet. Eftersom `JSON.stringify()` inte förändrar det ursprungliga indataobjektet, kommer detta objekt att förbli oförändrat i den behållare som gås igenom. Du kan således använda koden `this[k]` för att tillfråga nyckeln i originalobjektet. Parametern `v` innehåller värdet som konverteras med `toJSON()`.

Precis som `toJSON()`, kan funktionen `replacer` returnera olika typer av värden. Om `replacer` returnerar en sträng, kommer JSON-motorn att undertrycka innehållet med citattecken och omsluter sedan detta innehåll med ytterligare citattecken. Den här förpackningen garanterar att `stringify()` tar emot ett giltigt JSON-strängobjekt som behålls som en sträng i ett efterföljande anrop till `JSON.parse()`.

I följande kod används parametern `replacer` och den implicita `this`-parametern för att returnera värdena för `time` och `hours` i Date-objektet:

```
JSON.stringify(d, function (k,v):* {
    return "any date format you like via replacer: "+
        "holder[k].time:"+this[k].time + " holder[k].hours:"+this[k].hours;
});
```

Använda reviver-parametern i JSON.parse()

Parametern `reviver` i `JSON.parse()`-metoden gör det motsatta mot `replacer`-funktionen: Den konverterar en JSON-sträng till ett användbart ActionScript-objekt. Argumentet `reviver` är en funktion som tar två parametrar och returnerar valfri typ:

```
function (k,v):*
```

I denna funktion är `k` en nyckel och `v` värdet på `k`. Precis som `stringify()`, går `parse()` igenom nyckelvärdepar för JSON och tillämpar `reviver`-funktionen, om det finns någon, på varje par. Ett potentiellt problem är det faktum att JSON-klassen inte lämnar ut ett objekts ActionScript-klassnamn. Det kan således vara svårt att veta vilken typ av objekt som ska återställas. Detta problem kan vara speciellt besvärligt när objekten är kapslade. När du skapar `toJSON()`-, `replacer`- och `reviver`-funktioner, kan du göra egna identifiera för ActionScript-objekten som är exporterade, samtidigt som du behåller originalobjekten intakta.

Tolkningsexempel

I följande exempel visas en strategi för att återställa objekt tolkade från JSON-strängar. I det här exemplet definieras två klasser: JSONGenericDictExample och JSONDictionaryExtnExample. Klassen JSONGenericDictExample är en anpassad ordlisteklass. Varje post innehåller ett personnamn och födelsedatum samt ett unikt ID. Varje gång JSONGenericDictExample-konstruktorn anropas, lägger den till det nyligen skapade objektet i en intern statisk array med ett jämt ökande heltal som dess ID. Klassen JSONGenericDictExample definierar även en `revive()`-metod, som extraherar endast heltalsdelen från den längre `id`-medlemmen. Metoden `revive()` använder detta heltal för att söka efter och returnera rätt återställningsobjekt.

Klassen JSONDictionaryExtnExample utökar ActionScript-klassen Dictionary. Dess poster har ingen fast struktur, utan kan innehålla alla typer av data. Data tilldelas efter det att ett JSONDictionaryExtnExample-objekt har konstruerats, och inte som klassdefinierade egenskaper. JSONDictionaryExtnExample-poster använder JSONGenericDictExample-objekt som nycklar. När ett JSONDictionaryExtnExample-objekt återställs använder funktionen `JSONGenericDictExample.revive()` det ID som är associerat med JSONDictionaryExtnExample för att hämta korrekt nyckelobjekt.

Viktigast av allt är att metoden `JSONDictionaryExtnExample.toJSON()` returnerar en markörsträng, utöver JSONDictionaryExtnExample-objektet. Denna sträng identifierar att dessa JSON-utdata tillhör klassen JSONDictionaryExtnExample. Denna markör lämnar inget tvivel om vilken objekttyp som bearbetas med `JSON.parse()`.

```
package {
    // Generic dictionary example:
    public class JSONGenericDictExample {
        static var revivableObjects = [];
        static var nextId = 10000;
        public var id;
        public var dname:String;
        public var birthday;

        public function JSONGenericDictExample(name, birthday) {
            revivableObjects[nextId] = this;
            this.id       = "id_class_JSONGenericDictExample_" + nextId;
            this.dname     = name;
            this.birthday = birthday;
            nextId++;
        }
        public function toString():String { return this.dname; }
        public static function revive(id:String):JSONGenericDictExample {
            var r:RegExp = /^id_class_JSONGenericDictExample_([0-9]*)$/;
            var res = r.exec(id);
            return JSONGenericDictExample.revivableObjects[res[1]];
        }
    }
}

package {
    import flash.utils.Dictionary;
    import flash.utils.ByteArray;

    // For this extension of dictionary, we serialize the contents of the
    // dictionary by using toJSON
    public final class JSONDictionaryExtnExample extends Dictionary {
        public function toJSON(k):* {
            var contents = {};
```

```
            for (var a in this) {
                contents[a.id] = this[a];
            }

            // We also wrap the contents in an object so that we can
            // identify it by looking for the marking property "class E"
            // while in the midst of JSON.parse.
            return {"class JSONDictionaryExtnExample": contents};
        }

        // This is just here for debugging and for illustration
        public function toString():String {
            var retval = "[JSONDictionaryExtnExample <";
            var printed_any = false;
            for (var k in this) {
                retval += k.toString() + "=" +
                "[e="+this[k].earnings +
                ",v="+this[k].violations + "], "
                printed_any = true;
            }
            if (printed_any)
                retval = retval.substring(0, retval.length-2);
            retval += ">]"
            return retval;
        }
    }
}
```

När följande skript anropar `JSON.parse()` för ett JSONDictionaryExtnExample-objekt anropar funktionen `reviverJSONGenericDictExample.revive()` för alla objekt i JSONDictionaryExtnExample. Det här anropet extraherar det ID som representerar objektnyckeln. ID:t används sedan av funktionen `JSONGenericDictExample.revive()` för att hämta och returnera det lagrade JSONDictionaryExtnExample-objektet från en privat, statisk array.

```
import flash.display.MovieClip;
import flash.text.TextField;

var a_bob1:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("01/02/1934")));
var a_bob2:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("05/06/1978")));
var a_jen:JSONGenericDictExample = new JSONGenericDictExample("Jen", new
Date(Date.parse("09/09/1999")));

var e = new JSONDictionaryExtnExample();
e[a_bob1] = {earnings: 40, violations: 2};
e[a_bob2] = {earnings: 10, violations: 1};
e[a_jen]  = {earnings: 25, violations: 3};

trace("JSON.stringify(e): " + JSON.stringify(e)); // {"class JSONDictionaryExtnExample":
                        //{"id_class_JSONGenericDictExample_10001":
                        //{"earnings":10,"violations":1},
                        //"id_class_JSONGenericDictExample_10002":
                        //{"earnings":25,"violations":3},
                        //"id_class_JSONGenericDictExample_10000":
                        // {"earnings":40,"violations":2}}}
```

```
var e_result = JSON.stringify(e);

var e1 = new JSONDictionaryExtnExample();
var e2 = new JSONDictionaryExtnExample();

// It's somewhat easy to convert the string from JSON.stringify(e) back
// into a dictionary (turn it into an object via JSON.parse, then loop
// over that object's properties to construct a fresh dictionary).
//
// The harder exercise is to handle situations where the dictionaries
// themselves are nested in the object passed to JSON.stringify and
// thus does not occur at the topmost level of the resulting string.
//
// (For example: consider roundtripping something like
//   var tricky_array = [e1, [[4, e2, 6]], {table:e3}]
// where e1, e2, e3 are all dictionaries.  Furthermore, consider
// dictionaries that contain references to dictionaries.)
//
// This parsing (or at least some instances of it) can be done via
// JSON.parse, but it's not necessarily trivial.  Careful consideration
// of how toJSON, replacer, and reviver can work together is
// necessary.

var e_roundtrip =
    JSON.parse(e_result,
            // This is a reviver that is focused on rebuilding JSONDictionaryExtnExample objects.
                function (k, v) {
                     if ("class JSONDictionaryExtnExample" in v) { // special marker tag;
                         //see JSONDictionaryExtnExample.toJSON().
                        var e = new JSONDictionaryExtnExample();
                        var contents = v["class JSONDictionaryExtnExample"];
                        for (var i in contents) {
                           // Reviving JSONGenericDictExample objects from string
                           // identifiers is also special;
                           // see JSONGenericDictExample constructor and
                           // JSONGenericDictExample's revive() method.
                            e[JSONGenericDictExample.revive(i)] = contents[i];
                        }
                        return e;
                    } else {
                        return v;
```

```
                    }
                });

trace("// == Here is an extended Dictionary that has been round-tripped  ==");
trace("// == Note that we have revived Jen/Jan during the roundtrip.    ==");
trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jen=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jen=[e=25,v=3]>]
trace("Is e_roundtrip a JSONDictionaryExtnExample? " + (e_roundtrip is
JSONDictionaryExtnExample)); //true
trace("Name change: Jen is now Jan");
a_jen.dname = "Jan"

trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jan=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jan=[e=25,v=3]>]
```

Kapitel 8: Hantera händelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med hjälp av ett händelsehanteringssystem kan programmerare besvara användarinmatningar och systemhändelser på ett smidigt sätt. ActionScript 3.0-händelsemodellen är inte bara praktisk, utan följer även branschstandarder och är väl integrerad med visningslistan. Den nya händelsemodellen bygger på DOM Level 3-händelsespecifikationen, som är en standardarkitektur för händelsehantering. Modellen erbjuder ett effektivt och samtidigt intuitivt händelsehanteringsverktyg för ActionScript-programmerare.

Händelsehanteringssystemet i ActionScript 3.0 samverkar i hög grad med visningslistan. Läs ”Visningsprogrammering” på sidan 151 för att få en grundläggande förståelse för visningslistan.

Fler hjälpavsnitt

flash.events-paketet

DOM Level 3-händelsespecifikation

Grundläggande om händelsehantering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelser är olika typer av förekomster i den SWF-fil som du ska arbeta med som programmerare. De flesta SWF-filer stöder till exempel användarinteraktion av någon typ — vare sig det rör sig om något så enkelt som att svara med en musklickning, eller något mer avancerat som att godta och bearbeta data som har angetts i ett formulär. All sådan användarinteraktion med en SWF-fil anses vara en händelse. Händelser kan även inträffa utan någon användarinteraktion, som när data har lästs in från en server eller när en bifogad kamera har aktiverats.

I ActionScript 3.0 representeras varje händelse av ett händelseobjekt, som är en instans av klassen Event eller någon av dess underklasser. I ett händelseobjekt sparas inte bara information om en specifik händelse. Objektet innehåller även metoder som underlättar hanteringen av händelseobjektet. När Flash Player eller AIR identifierar en musklickning, skapas ett händelseobjekt (en instans av klassen MouseEvent) som ska representera den specifika musklickningshändelsen.

När ett händelseobjekt har skapats *skickar* Flash Player eller AIR objektet, vilket innebär att händelseobjektet överförs till objektet som är målet för händelsen. Ett objekt som fungerar som mål för ett skickat händelseobjekt kallas för ett *händelsemål*. När till exempel en bifogad kamera aktiveras, skickar Flash Player ett händelseobjekt direkt till händelsemålet, som i det här fallet är objektet som representerar kameran. Om händelsemålet däremot finns i visningslistan, överförs händelseobjektet nedåt i visningslistans hierarki tills det når händelsemålet. I vissa fall ”bubblar” sedan händelseobjekt tillbaka uppåt i visningslistans hierarki längs samma väg. Den här förflyttningen eller gången i visningslistans hierarki kallas för *händelseflödet*.

Du kan ”lyssna” efter händelseobjekt i kod med hjälp av händelseavlyssnare. *Händelseavlyssnare* är funktioner eller metoder som svarar på specifika händelser. För att se till att ditt program svarar på händelser måste du lägga till händelseavlyssnare till antingen händelsemålet eller till ett visningslisteobjekt som ingår i ett händelseobjekts händelseflöde.

Varje gång du skriver händelseavlyssnarkod har koden följande grundläggande struktur (element i fet stil är platshållare som du fyller i med information som är lämplig i ditt specifika fall):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

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

Den här koden gör två saker. Först definierar den en funktion, vilket anger åtgärderna som ska utföras som svar på händelsen. Därefter anropar den metoden `addEventListener()` för källobjektet, vilket kopplar funktionen till den angivna händelsen. Därmed utförs funktionens åtgärder varje gång händelsen inträffar. När händelsen verkligen inträffar, kontrollerar händelsemålet dess lista över alla funktioner och metoder som är registrerade som händelseavlyssnare. Därefter anropas var och en i tur och ordning och händelseobjektet skickas som en parameter.

Du måste ändra fyra saker i den här koden för att skapa din egen händelseavlyssnare. Först måste du ändra namnet på funktionen till det namn som du vill använda (namnet måste ändras på två platser, där koden anger **eventResponse**). Sedan måste du ange lämpligt klassnamn för händelseobjektet som skickas av händelsen som du vill lyssna efter (**EventType** i koden) och ange lämplig konstant för den specifika händelsen (**EVENT_NAME** i exemplet). Det tredje steget innebär att du anropar metoden `addEventListener()` i objektet som ska skicka händelsen (**eventTarget** i den här koden). Om du vill kan du som ett fjärde steg ändra namnet på variabeln som används som funktionens parameter (**eventObject** i den här koden).

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kommer att stöta på när du skriver händelsehanteringsrutiner:

Bubbling Bubbling sker för vissa händelser så att ett överordnat visningsobjekt kan svara på händelser som skickas av dess underordnade objekt.

Bubblingsfas Den del av händelseflödet i vilken en händelse sprids uppåt till de överordnade visningsobjekten. Bubblingsfasen inträffar efter hämtnings- och målfaserna.

Hämtningsfas Den del av händelseflödet i vilken en händelse sprids nedåt från det mest allmänna målet till det mest specifika målobjektet. Hämtningsfasen inträffar före mål- och bubblingsfaserna.

Standardbeteende Vissa händelser innehåller ett beteende som normalt inträffar tillsammans med händelsen. Detta kallas standardbeteende. När användaren till exempel skriver text i ett textfält, kan en textinmatningshändelse utlösas. Standardbeteendet för den händelsen är att visa tecknet som skrevs i textfältet, men du kan åsidosätta standardbeteendet (om du av någon anledning inte vill att det skrivna tecknet ska visas).

Skicka Att meddela händelseavlyssnare om att en händelse har inträffat.

Händelse Något som sker med ett objekt och som objektet kan informera andra objekt om.

Händelseflöde När händelser inträffar för ett objekt i visningslistan (ett objekt som visas på skärmen) meddelas alla objekt som innehåller objektet om händelsen. De objekten meddelar sedan i sin tur sina händelseavlyssnare. Den här processen börjar med scenen och fortsätter sedan igenom visningslistan till det faktiska objekt där händelsen inträffade. Sedan fortsätter processen tillbaka till scenen igen. Processen kallas även för händelseflödet.

Händelseobjekt Ett objekt som innehåller information om en viss händelses förekomst, som skickas till alla avlyssnare när en händelse skickas.

Händelsemål Det objekt som verkligen skickar en händelse. Om användaren till exempel klickar på en knapp som finns i ett sprite-objekt, som i sin tur finns på scenen, skickar alla de objekten händelser, men händelsemålet är den där händelsen faktiskt inträffade (den klickade knappen i det här fallet).

Lyssnare Ett objekt eller en funktion som har registrerat sig själv med ett objekt för att ange att det eller den vill meddelas när en viss händelse inträffar.

Målfas Den punkt i ett händelseflöde där en händelse har nått det mest specifika mål som är möjligt. Målfasen inträffar mellan hämtnings- och bubblingsfaserna.

Hur ActionScript 3.0-händelsehantering skiljer sig från tidigare versioner

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den mest utmärkande skillnaden mellan händelsehantering i ActionScript 3.0 och händelsehantering i tidigare versioner av ActionScript är att det i ActionScript 3.0 bara finns ett system för händelsehantering. I tidigare versioner av ActionScript finns det flera händelsehanteringssystem. Det här avsnittet börjar med en översikt över hur händelsehantering fungerade i tidigare versioner av ActionScript. Därefter beskrivs hur händelsehantering har ändrats för ActionScript 3.0.

Händelsehantering i tidigare versioner av ActionScript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript-versioner före ActionScript 3.0 erbjuder en rad olika sätt att hantera händelser på, bland annat följande:

- `on()`-händelsehanterare som kan placeras direkt i Button- och MovieClip-instanser
- `onClipEvent()`-hanterare som kan placeras direkt i MovieClip-instanser
- Återkallningsfunktionsegenskaper som `XML.onload` och `Camera.onActivity`
- Händelseavlyssnare som du registrerar med hjälp av metoden `addListener()`
- Klassen UIEventDispatcher som delvis implementerade DOM-händelsemodellen.

Var och en av dessa mekanismer har sin egen uppsättning fördelar och begränsningar. Hanterarna `on()` och `onClipEvent()` är enkla att använda, men gör senare underhåll av projektet svårare eftersom kod som har placerats direkt i knappar och filmklipp kan vara svåra att hitta. Återkallningsfunktioner är också enkla att implementera, men begränsar dig så att du bara kan använda en återkallningsfunktion för en given händelse. Händelseavlyssnare är svårare att implementera eftersom de inte bara kräver att ett avlyssnarobjekt och en avlyssnarfunktion skapas, utan även kräver att avlyssnaren registreras med objektet som genererar händelsen. Det här extraarbetet gör dock att du kan skapa flera avlyssnarobjekt och registrera dem alla för samma händelse.

Utvecklingen av komponenter för ActionScript 2.0 gav upphov till ytterligare en händelsemodell. Den nya modellen, i form av klassen UIEventDispatcher, grundades på en deluppsättning av DOM-händelsespecifikationen. Utvecklare som är insatta i händelsehantering kommer därför inte att ha några större problem att gå över till den nya ActionScript 3.0-händelsemodellen.

Tyvärr sammanfaller syntaxen som används av de olika händelsemodellerna på olika sätt, och skiljer sig åt på andra sätt. I ActionScript 2.0 kan till exempel vissa egenskaper, som `TextField.onChanged`, användas antingen som en återkallningsfunktion eller som en händelseavlyssnare. Syntaxen för att registrera avlyssnarobjekt skiljer sig däremot åt beroende på om du använder någon av de sex klasser som stöder avlyssnare eller klassen UIEventDispatcher. För klasserna Key, Mouse, MovieClipLoader, Selection, Stage och TextField använder du metoden `addListener()`, men för komponenthändelsehantering använder du en metod som heter `addEventListener()`.

En annan svårighet som de olika händelsehanteringsmodellerna introducerade var att omfånget på händelsehanteringsfunktionen varierade avsevärt beroende på vilken mekanism som användes. Innebörden av nyckelordet `this` var med andra ord inte konsekvent mellan de olika händelsehanteringssystemen.

Händelsehantering i ActionScript 3.0

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript 3.0 introducerar en enda händelsehanteringsmodell som ersätter de många olika händelsehanteringsmekanismerna som fanns i tidigare versioner av språket. Den nya händelsemodellen bygger på Document Object Model (DOM) Level 3-händelsespecifikationen. Även om SWF-filformatet inte följer DOM-standarden specifikt, finns det tillräckligt med likheter mellan visningslistan och DOM-strukturen för att göra det möjligt att implementera DOM-händelsemodellen. Ett objekt i visningslistan är detsamma som en nod i DOM-hierarkistrukturen, och termerna *visningslisteobjekt* och *nod* används båda för att beteckna samma sak.

Flash Player- och AIR-implementeringen av DOM-händelsemodellen innefattar begreppet standardbeteende. Ett *standardbeteende* är en åtgärd som Flash Player eller AIR utför som standard i samband med vissa händelser.

Standardbeteenden

Utvecklare har normalt ansvar för att skriva kod som svarar på händelser. I vissa fall är dock ett beteende så självklart kopplat till en händelse att Flash Player eller AIR automatiskt utför det beteendet, såvida inte utvecklaren lägger till kod som upphäver det. Eftersom Flash Player eller AIR uppvisar beteendet automatiskt kallas sådana beteenden standardbeteenden.

Om en användare till exempel anger text i ett TextField-objekt är det så självklart att texten förväntas visas i det TextField-objektet att detta beteende har byggts in i Flash Player och AIR. Om du inte vill att detta standardbeteende ska gälla, kan du avbryta det genom att använda det nya händelsehanteringssystemet. När en användare anger text i ett TextField-objekt, skapar Flash Player eller AIR en instans av klassen TextEvent som ska representera användarens indata. För att förhindra att Flash Player eller AIR visar texten i TextField-objektet måste du komma åt den specifika TextEvent-instansen och anropa den instansens `preventDefault()`-metod.

Vissa standardbeteenden går inte att förhindra. Flash Player och AIR genererar till exempel ett MouseEvent-objekt när en användare dubbelklickar på ett ord i ett TextField-objekt. Standardbeteendet, som inte går att förhindra, är att ordet under markören markeras.

Många typer av händelseobjekt saknar standardbeteenden. Flash Player skickar till exempel ett connect-händelseobjekt när en nätverksanslutning upprättas, men det finns inget standardbeteende kopplat till objektet. I API-dokumentationen för klassen Event och dess underklasser beskrivs alla typer av händelser och deras associerade standardbeteenden, och den innehåller även information om huruvida ett beteende går att förhindra eller inte.

Det är viktigt att veta att standardbeteenden bara är kopplade till händelseobjekt som skickas av Flash Player eller AIR och att de inte existerar för händelseobjekt som skickas med programkod via ActionScript. Du kan till exempel använda metoderna för klassen EventDispatcher för att skicka ett händelseobjekt av typen `textInput`, men det kommer inte att finnas något standardbeteende kopplat till det händelseobjektet. Flash Player kommer med andra ord inte att visa ett tecken i ett TextField-objekt som ett resultat av en `textInput`-händelse som du skickade med programkod.

Nyheter för händelseavlyssnare i ActionScript 3.0

För utvecklare med erfarenhet av att använda metoden `addListener()` i ActionScript 2.0 kan det vara till hjälp att nämna skillnaderna mellan händelseavlyssnarmodellen i ActionScript 2.0 och händelsemodellen i ActionScript 3.0. I listan nedan beskrivs några av de viktigaste skillnaderna mellan de båda händelsemodellerna:

- För att lägga till händelseavlyssnare i ActionScript 2.0 använder du `addListener()` i vissa fall och `addEventListener()` i andra. I ActionScript 3.0 använder du däremot `addEventListener()` i alla situationer.
- Det finns inget händelseflöde i ActionScript 2.0, vilket innebär att metoden `addListener()` bara kan anropas i objektet som sänder händelsen. I ActionScript 3.0 kan däremot metoden `addEventListener()` anropas i valfritt objekt som ingår i händelseflödet.
- I ActionScript 2.0 kan händelseavlyssnare vara funktioner, metoder eller objekt. I ActionScript 3.0 kan bara funktioner och metoder vara händelseavlyssnare.

Händelseflödet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

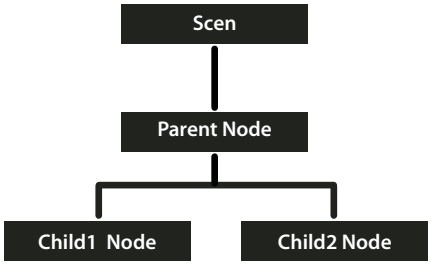
Flash Player och AIR skickar händelseobjekt varje gång en händelse inträffar. Om händelsemålet inte finns med i visningslistan, skickar Flash Player eller AIR händelseobjektet direkt till händelsemålet. Flash Player skickar till exempel progress-händelseobjektet direkt till ett URLStream-objekt. Om händelsemålet däremot finns med i visningslistan, skickar Flash Player händelseobjektet till visningslistan och sedan förflyttar sig händelseobjektet igenom visningslistan till händelsemålet.

Händelseflödet beskriver hur ett händelseobjekt förflyttar sig i visningslistan. Visningslistan är ordnad i en hierarki ungefär som ett träd. Överst i visningslistans hierarki finns scenen, som är en särskild behållare för visningsobjekt och fungerar som visningslistans rot. Scenen representeras av klassen flash.display.Stage och går bara att komma åt via ett visningsobjekt. Varje visningsobjekt har en egenskap med namnet `stage` som refererar till scenen för det programmet.

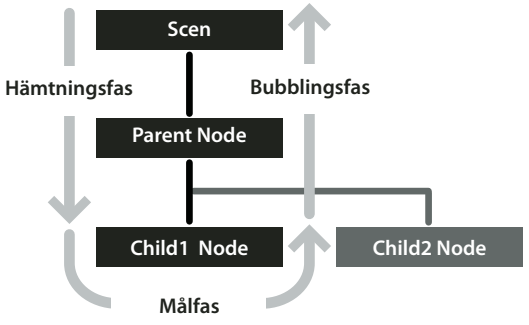
När Flash Player eller AIR skickar ett händelseobjekt för en händelse som rör visningslistan gör händelseobjektet en rundresa från scenen till *målnoden*. DOM-händelsespecifikationen definierar målnoden som den nod som representerar händelsemålet. Målnoden är med andra ord visningslisteobjektet där händelsen inträffade. Om en användare till exempel klickar på ett visningslisteobjekt med namnet `child1`, skickar Flash Player eller AIR ett händelseobjekt och använder `child1` som målnod.

Händelseflödet är begreppsmässigt indelat i tre delar. Den första delen kallas för hämtningsfasen och består av alla noder från scenen till målnodens överordnade nod. Den andra delen kallas för målfasen och består enbart av målnoden. Den tredje delen kallas för bubblingsfasen. Bubblingsfasen består av noderna som påträffades på tillbakavägen från målnodens överordnade nod till scenen.

Namnen på faserna blir mer logiska om du tänker på visningslistan som en lodrät hierarki med scenen högst upp som i följande diagram:



Om en användare klickar på `Child1 Node` skickar Flash Player eller AIR ett händelseobjekt till händelseflödet. Som nedanstående bild visar börjar objektets resa vid `Stage`, rör sig nedåt till `Parent Node`, vidare till `Child1 Node` och ”bubblar” sedan tillbaka till `Stage` genom att passera genom `Parent Node` igen på vägen tillbaka till `Stage`.



I det här exemplet utgörs hämtningsfasen av `Stage` och `Parent Node` under den inledande resan nedåt. Målfasen utgörs av tiden vid `Child1 Node`. Bubblingsfasen utgörs av `Parent Node` och `Stage` när de påträffas under resan tillbaka uppåt till rotnoden.

Händelseflödet ger ett mer effektivt händelsehanteringssystem än det som tidigare fanns tillgängligt för ActionScript-programmerare. I tidigare versioner av ActionScript finns inte händelseflödet, vilket gör att händelseavlyssnare bara kan läggas in i objektet som genererar händelsen. I ActionScript 3.0 kan du lägga till händelseavlyssnare till valfri nod i händelseflödet och inte bara till målnoden.

Möjligheten att kunna lägga till händelseavlyssnare längs händelseflödet är användbart när en komponent i användargränssnittet består av flera objekt. Ett knappobjekt består till exempel ofta av ett textobjekt som fungerar som knappens etikett. Utan möjligheten att kunna lägga till en avlyssnare till händelseflödet skulle du behöva lägga till en avlyssnare till både knappobjektet och textobjektet för att se till att du meddelas om klickningshändelser som inträffar på knappen. Händelseflödet gör dock att du kan placera en enda händelseavlyssnare i knappobjektet som hanterar klickningshändelser som inträffar i textobjektet eller de områden av knappobjektet som inte är dolda av textobjektet.

Det finns vissa händelseobjekt som inte deltar i alla tre faserna av händelseflödet. Vissa typer av händelser, som `enterFrame` och `init`, skickas direkt till målnoden och deltar inte i varken hämtningsfasen eller bubblingsfasen. Andra händelser kan ha målobjekt som inte finns med i visningslistan, till exempel händelser som skickas till en instans av klassen Socket. Sådana händelseobjekt går även direkt till målobjektet utan att delta i hämtnings- och bubblingsfaserna.

För att ta reda på hur en viss typ av händelse fungerar kan du antingen läsa i API-dokumentationen eller undersöka händelseobjektets egenskaper. Hur du undersöker händelseobjektets egenskaper beskrivs i nästa avsnitt.

Händelseobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelseobjekt har två huvuduppgifter i det nya händelsehanteringssystemet. För det första representerar händelseobjekt faktiska händelser genom att lagra information om specifika händelser i en uppsättning egenskaper. För det andra innehåller händelseobjekt en uppsättning metoder som gör att du kan hantera händelseobjekt och påverka hur händelsehanteringssystemet fungerar.

För att underlätta tillgången till dessa egenskaper och metoder innehåller Flash Player-gränssnittet en klass, Event, som fungerar som basklass för alla händelseobjekt. Klassen Event definierar en grundläggande uppsättning egenskaper och metoder, som är gemensamma för alla händelseobjekt.

Det här avsnittet inleds med information om egenskaperna hos klassen Event, fortsätter med en beskrivning av klassens metoder och slutar med en förklaring av varför klassen finns.

Egenskaper för klassen Event

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Event definierar en antal skrivskyddade egenskaper och konstanter som erbjuder viktig information om ett händelseobjekt. Följande är särskilt viktigt:

- Händelseobjekttyper representeras av konstanter och lagras i egenskapen `Event.type`.
- Huruvida en händelses standardbeteende kan förhindras representeras av ett booleskt värde och lagras i egenskapen `Event.cancelable`.
- Händelseflödesinformation lagras i de övriga egenskaperna.

Typer av händelseobjekt

Alla händelseobjekt har en tillhörande händelsetyp. Händelsetyper lagras i egenskapen `Event.type` som strängvärden. Den är användbar för att ta reda på ett händelseobjekts typ så att din kod kan särskilja objekt av olika typer. Följande kod anger till exempel att avlyssnarfunktionen `clickHandler()` bör svara på händelseobjekt av typen musklickning som skickas till `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Ett par dussin händelsetyper är kopplade till själva klassen Event och representeras av konstanter för klassen. Några av de konstanterna visas i följande utdrag från definitionen av klassen Event:

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

Dessa konstanter underlättar när du vill referera till specifika händelsetyper. Du bör använda dessa konstanter i stället för strängarna som de representerar. Om du stavar ett konstantnamn fel i din kod kommer kompilatorn att fånga felet. Om du däremot använder strängar kan skrivfel inte framkomma vid kompileringen och kan därför leda till fel som kan vara svåra att hitta. Använd till exempel följande kod när du lägger till en händelseavlyssnare:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

i stället för:

```
myDisplayObject.addEventListener("click", clickHandler);
```

Information om standardbeteende

Koden kan kontrollera om standardbeteendet för ett givet händelseobjekt kan förhindras genom att använda egenskapen `cancelable`. Egenskapen `cancelable` innehåller ett booleskt värde som anger om ett standardbeteende kan förhindras eller inte. Du kan förhindra eller avbryta standardbeteendet som är kopplat till ett litet antal händelser genom att använda metoden `preventDefault()`. Mer information finns i Avbryta standardbeteende för händelse under ”Metoder för klassen Event” på sidan 133.

Information om händelseflöde

De återstående egenskaperna för klassen Event innehåller viktig information om ett händelseobjekt och dess relation till händelseflödet enligt beskrivningen i följande lista:

- Egenskapen `bubbles` innehåller information om de delar av händelseflödet som händelseobjektet deltar i.
- Egenskapen `eventPhase` indikerar aktuell fas i händelseflödet.
- Egenskapen `target` lagrar en referens till händelsemålet.
- Egenskapen `currentTarget` lagrar en referens till visningslisteobjektet som för närvarande bearbetar händelseobjektet.

Egenskapen bubbles

En händelse sägs bubbla om dess händelseobjekt deltar i bubblingsfasen i händelseflödet, vilket innebär att händelseobjektet överförs från målnoden tillbaka genom dess indirekta överordnade tills det når scenen. Egenskapen `Event.bubbles` lagrar ett booleskt värde som indikerar om händelseobjektet deltar i bubblingsfasen. Eftersom alla händelser som bubblar även deltar i hämtnings- och målfaserna, deltar alla händelser som bubblar i alla tre av händelseflödets faser. Om värdet är `true` deltar händelseobjektet i alla tre faser. Om värdet är `false` deltar händelseobjektet inte i bubblingsfasen.

Egenskapen eventPhase

Du kan bestämma händelsefasen för ett händelseobjekt genom att undersöka dess `eventPhase`-egenskap. Egenskapen `eventPhase` innehåller ett heltalsvärde utan tecken som representerar en av de tre faserna i händelseflödet. Flash Player API innehåller en separat EventPhase-klass med tre konstanter som motsvarar de tre heltalsvärdena utan tecken som i följande kodutdrag:

```
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;
    }
}
```

Dessa konstanter motsvarar de tre giltiga värdena för egenskapen `eventPhase`. Du kan använda dessa konstanter för att göra din kod enklare att läsa. Om du till exempel vill se till att en funktion med namnet `myFunc()` bara anropas om händelsemålet är i målfasen, kan du använda följande kod för att testa detta:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

Egenskapen target

Egenskapen `target` innehåller en referens till objektet som är målet för händelsen. I vissa fall är detta relativt okomplicerat, såsom när en mikrofon blir aktiv då Microphone-objektet blir målet för händelseobjektet. Om målet däremot finns i visningslistan måste visningslistans hierarki tas med i beräkningen. Om en användare till exempel klickar med musen på en punkt som inkluderar överlappande visningslisteobjekt, väljer Flash Player och AIR alltid objektet som är längst bort från scenen som händelsemål.

För invecklade SWF-filer, särskilt sådana i vilka knappar vanligtvis förses med mindre underordnade objekt, används egenskapen `target` normalt inte så mycket eftersom den ofta kommer att peka på en knapps underordnade objekt i stället för på knappen. I så fall är det brukligt att lägga till händelseavlyssnare till knappen och att sedan använda egenskapen `currentTarget` eftersom den pekar på knappen. Egenskapen `target` kan däremot peka på ett av knappens underordnade objekt.

Egenskapen currentTarget

Egenskapen `currentTarget` innehåller en referens till det objekt som för närvarande bearbetar händelseobjektet. Även om det kan förefalla märkligt att inte känna till vilken nod som för närvarande bearbetar händelseobjektet som du undersöker, bör du komma ihåg att du kan lägga en avlyssnarfunktion till valfritt visningsobjekt i det händelseobjektets händelseflöde, och att avlyssnarfunktionen kan placeras på valfri plats. Det går dessutom att lägga till samma avlyssnarfunktion till olika visningsobjekt. Allteftersom projektet blir större och mer invecklat, blir egenskapen `currentTarget` mer och mer användbar.

Metoder för klassen Event

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns tre kategorier med metoder för klassen Event:

- Verktygsmetoder, som kan skapa kopior av ett händelseobjekt eller konvertera det till en sträng
- Händelseflödesmetoder, som tar bort händelseobjekt från händelseflödet
- Standardbeteendemetoder, som förhindrar standardbeteende eller kontrollerar om beteendet har förhindrats

Verktygsmetoder i klassen Event

Det finns två verktygsmetoder i klassen Event. Metoden `clone()` gör att du kan skapa kopior av ett händelseobjekt. Med metoden `toString()` kan du generera en strängrepresentation av ett händelseobjekts egenskaper tillsammans med deras värden. Båda dessa metoder används internt av händelsemodellsystemet, men är tillgängliga för programmerare att använda.

För erfarna programmerare som skapar underklasser till klassen Event måste du åsidosätta och implementera versioner av båda verktygsmetoderna för att händelseunderklassen ska fungera.

Stoppa ett händelseflöde

Du kan anropa antingen metoden `Event.stopPropagation()` eller metoden `Event.stopImmediatePropagation()` om du vill förhindra att ett händelseobjekt fortsätter genom händelseflödet. De båda metoderna är nästan identiska och skiljer sig bara åt när det gäller huruvida den aktuella nodens övriga händelseavlyssnare får köras:

- Metoden `Event.stopPropagation()` förhindrar att händelseobjektet fortsätter till nästa nod, men bara efter att eventuella övriga händelseavlyssnare i aktuell nod tillåts köras.
- Metoden `Event.stopImmediatePropagation()` förhindrar också att händelseobjektet fortsätter till nästa nod, men tillåter inte att några andra händelseavlyssnare i aktuell nod körs.

Anropande av någon av dessa metoder påverkar inte om standardbeteendet som är kopplat till en händelse utförs. Använd standardbeteendemetoderna i klassen Event för att förhindra standardbeteende.

Avbryta standardbeteende för händelse

De två metoder som finns för att avbryta standardbeteendet är `preventDefault()` och `isDefaultPrevented()`. Anropa metoden `preventDefault()` om du vill avbryta standardbeteendet som är kopplat till en händelse. Om du vill kontrollera om `preventDefault()` redan har anropats för ett händelseobjekt, anropar du metoden `isDefaultPrevented()`. Värdet `true` returneras om metoden redan har anropats. Annars returneras värdet `false`.

Metoden `preventDefault()` fungerar bara om händelsens standardbeteende går att avbryta. Du kan kontrollera om så är fallet genom att läsa i API-dokumentationen för den händelsetypen, eller genom att använda ActionScript för att undersöka `cancelable`-egenskapen för händelseobjektet.

Om du avbryter standardbeteendet har detta ingen effekt på händelseobjektets förlopp genom händelseflödet. Använd händelseflödesmetoden för klassen Event om du vill ta bort ett händelseobjekt från händelseflödet.

Klassen Events underklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För många händelser är den vanliga uppsättningen med egenskaper som är definierade i klassen Event fullt tillräcklig. Andra händelser kan dock ha unika särdrag som inte täcks av de egenskaper som är tillgängliga i klassen Event. För dessa händelser definierar ActionScript 3.0 flera underklasser till klassen Event.

Varje underklass har ytterligare egenskaper och händelsetyper, som är unika för den kategorin av händelser. Händelser som rör musklickning har till exempel flera unika särdrag som inte täcks av egenskaperna som är definierade i klassen Event. Klassen MouseEvent utökar klassen Event genom att lägga till tio egenskaper. Dessa innehåller information om till exempel platsen för mushändelsen och om huruvida särskilda tangenttryckningar gjordes under mushändelsen.

En Event-underklass innehåller även konstanter som representerar händelsetyperna som är kopplade till underklassen. Klassen MouseEvent definierar till exempel konstanter för flera mushändelsetyper, inklusive händelsetyperna `click`, `doubleClick`, `mouseDown` och `mouseUp`.

Enligt beskrivningen i avsnittet Verktygsmetoder i klassen Event under ”Händelseobjekt” på sidan 130 måste du, när du skapar en Event-underklass, åsidosätta metoderna `clone()` och `toString()` för att erbjuda funktionalitet som är specifik för underklassen.

Händelseavlyssnare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelseavlyssnare, som även kallas händelsehanterare, är funktioner som Flash Player och AIR kör som svar på olika händelser. Du lägger till en händelseavlyssnare i två steg. Först skapar du en funktion eller klassmetod som Flash Player eller AIR ska köra som svar på en händelse. Detta kallas ibland för avlyssnarfunktionen eller händelsehanterarfunktionen. Sedan använder du metoden `addEventListener()` för att registrera avlyssnarfunktionen med målet för händelsen eller med något visningslisteobjekt som finns i lämpligt händelseflöde.

Skapa en avlyssnarfunktion

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Sättet på vilket avlyssnarfunktioner skapas i ActionScript 3.0-händelsemodellen uppvisar vissa avvikelser från DOM-händelsemodellen. I DOM-händelsemodellen finns det en klar skillnad mellan en händelseavlyssnare och en avlyssnarfunktion: en händelseavlyssnare är en instans av en klass som implementerar EventListener-gränssnittet, medan en avlyssnarfunktion är en metod i den klassen med namnet `handleEvent()`. I DOM-händelsemodellen registrerar du klassinstansen som innehåller avlyssnarfunktionen snarare än den faktiska avlyssnarfunktionen.

I ActionScript 3.0-händelsemodellen är det ingen skillnad på en händelseavlyssnare och en avlyssnarfunktion. ActionScript 3.0 har inte något EventListener-gränssnitt, och avlyssnarfunktioner kan definieras utanför en klass eller som del av en klass. Dessutom behöver avlyssnarfunktioner inte heta `handleEvent()`. De kan namnges med valfri, giltig identifierare. I ActionScript 3.0 registrerar du namnet på den faktiska avlyssnarfunktionen.

Avlyssnarfunktion definierad utanför en klass

Nedanstående kod skapar en enkel SWF-fil som visar en röd kvadrat. En avlyssnare med namnet `clickHandler()`, som inte ingår i någon klass, lyssnar efter musklickningshändelser som sker i den röda rutan.

```
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);
}
```

När en användare interagerar med den slutliga SWF-filen genom att klicka på rutan, genererar Flash Player eller AIR följande spårningsutdata:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Observera att händelseobjektet överförs som ett argument till `clickHandler()`. Detta gör att din avlyssnarfunktion kan undersöka händelseobjektet. I det här exemplet använder du händelseobjektets `type`-egenskap för att kontrollera om händelsen är en klickningshändelse.

Exemplet kontrollerar även värdet för nyckelordet `this`. I det här fallet representerar `this` det globala objektet, vilket är rimligt eftersom funktionen har definierats utanför en anpassad klass eller objekt.

Avlyssnarfunktion definierad som en klassmetod

Följande exempel är identiskt med föregående exempel som definierar klassen ClickExample, men här definieras funktionen `clickHandler()` som en metod i klassen ChildSprite:

```
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);
    }
}
```

När en användare interagerar med den slutliga SWF-filen genom att klicka på den röda rutan, genererar Flash Player eller AIR följande spårningsutdata:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

Observera att nyckelordet `this` refererar till ChildSprite-instansen med namnet `child`. Detta beteende skiljer sig från det i ActionScript 2.0. Om du använde komponenter i ActionScript 2.0 kanske du kommer ihåg att när en klassmetod överfördes till `UIEventDispatcher.addEventListener()`, kopplades omfånget på metoden till komponenten som sände händelsen i stället för till klassen i vilken avlyssnarmetoden definierades. Så om du använde den här tekniken i ActionScript 2.0 skulle nyckelordet `this` referera till komponenten som sänder händelsen i stället för till ChildSprite-instansen.

Det här var till nackdel för vissa programmerare, eftersom det innebar att de inte kunde få tillgång till andra metoder eller egenskaper för klassen som innehöll avlyssnarmetoden. För att komma runt problemet kunde ActionScript 2.0-programmerare använda klassen `mx.util.Delegate` för att ändra omfånget på avlyssnarmetoden. Detta är nu inte längre nödvändigt, eftersom ActionScript 3.0 skapar en bunden metod när `addEventListener()` anropas. Det innebär att nyckelordet `this` refererar till ChildSprite-instansen med namnet `child`, och programmeraren får tillgång till de andra metoderna och egenskaperna i klassen ChildSprite.

Händelseavlyssnare som inte bör användas

Det finns en tredje teknik som kan användas för att skapa ett generiskt objekt med en egenskap som pekar på en dynamiskt tilldelad avlyssnarfunktion, men det här är inte en teknik som rekommenderas. Den beskrivs bara här för att den ofta användes i ActionScript 2.0. Den bör inte användas i ActionScript 3.0. Den här tekniken rekommenderas inte eftersom nyckelordet `this` kommer att referera till det globala objektet i stället för till avlyssnarobjektet.

Följande exempel är identiskt med föregående exempel från klassen ClickExample förutom att avlyssnarfunktionen definieras som en del av ett generiskt objekt med namnet `myListenerObj`:

```
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);
}
```

Resultatet av spårningen kommer att se ut så här:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Man skulle kunna tro att `this` refererar till `myListenerObj` och att spårningsutdata är `[object Object]`, med det refererar i stället till det globala objektet. När du anger namnet på en dynamisk egenskap som argument till `addEventListener()`, kan Flash Player eller AIR inte skapa någon bunden metod. Det beror på att det du anger som `listener`-parameter inte är något annat än minnesadressen för avlyssnarfunktionen, och Flash Player och AIR kan inte länka den minnesadressen till `myListenerObj`-instansen .

Hantera händelseavlyssnare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan hantera dina avlyssnarfunktioner med hjälp av metoderna för IEventDispatcher-gränssnittet. IEventDispatcher-gränssnittet är ActionScript 3.0-versionen av EventTarget-gränssnittet i DOM-händelsemodellen. Även om namnet IEventDispatcher verkar antyda att dess främsta funktion är att skicka händelseobjekt, används metoderna i den här klassen faktiskt mer för att registrera, kontrollera och ta bort händelseavlyssnare. IEventDispatcher-gränssnittet definierar fem metoder, som i följande kod:

```
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 API implementerar IEventDispatcher-gränssnittet med klassen EventDispatcher, som fungerar som basklass för alla klasser som kan vara händelsemål eller del av ett händelseflöde. Klassen DisplayObject ärver till exempel från klassen EventDispatcher. Det innebär att objekt i visningslistan har tillgång till metoderna i IEventDispatcher-gränssnittet.

Lägga till händelseavlyssnare

`addEventListener()`-metoden är arbetshästen hos IEventDispatcher-gränssnittet. Du använder den för att registrera avlyssnarfunktioner. De två parametrarna som krävs är `type` och `listener`. Du använder parametern `type` för att ange typen av händelse. Du använder parametern `listener` för att ange avlyssnarfunktionen som ska köras när händelsen inträffar. Parametern `listener` kan vara en referens till antingen en funktion eller en klassmetod.

Använd inte parenteser när du anger `listener`-parametern. Funktionen `clickHandler()` anges till exempel utan parentes i följande anrop till `addEventListener()`-metoden:

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

Parametern `useCapture` för metoden `addEventListener()` gör att du kan styra händelseflödesfasen som avlyssnaren ska vara aktiv för. Om `useCapture` är `true` kommer avlyssnaren att vara aktiv under händelseflödets hämtningsfas. Om `useCapture` är `false` kommer avlyssnaren att vara aktiv under händelseflödets målfas och bubblingsfas. Om du vill lyssna efter en händelse under händelseflödets alla tre faser måste du anropa `addEventListener()` två gånger, en gång med `useCapture` inställd på `true` och sedan en gång till med `useCapture` inställd på `false`.

Parametern `priority` för metoden `addEventListener()` är inte en officiell del av DOM Level 3-händelsemodellen. Den ingår i ActionScript 3.0 för att du på ett mer flexibelt sätt ska kunna hantera dina händelseavlyssnare. När du anropar `addEventListener()` kan du ställa in prioriteten för den händelseavlyssnaren genom att ange ett heltalsvärde som `priority`-parameter. Standardvärdet är 0, men du kan ställa in den på negativa eller positiva heltalsvärden. Ju högre värde, desto tidigare kommer händelseavlyssnaren att köras. Händelseavlyssnare som har samma prioritet körs i den ordning de infogades, så ju tidigare en avlyssnare läggs till, desto tidigare kommer den att köras.

Parametern `useWeakReference` gör att du kan ange om referensen till avlyssnarfunktionen är svag eller normal. Om du ställer in den här parametern på `true` kan du undvika situationer där avlyssnarfunktioner finns kvar i minnet även när de inte längre behövs. Flash Player och AIR använder en teknik som kallas *skräpsamling* för att rensa de objekt från minnet som inte längre används. Ett objekt anses inte längre vara i bruk om det inte finns någon referens till objektet. Skräpsamlingen ignorerar svaga referenser, vilket innebär att en avlyssnarfunktion som bara har en svag referenspunkt som pekar på den kvalificerar sig för skräpsamling.

Ta bort händelseavlyssnare

Du kan använda metoden `removeEventListener()` när du vill ta bort en händelseavlyssnare som inte längre behövs. Det kan vara en bra idé att ta bort avlyssnare som inte längre används. Obligatoriska parametrar omfattar parametrarna `eventName` och `listener`, som är desamma som de parametrar som krävs för metoden `addEventListener()`. Kom ihåg att du kan lyssna efter händelser under alla händelsefaser genom att anropa `addEventListener()` två gånger, en gång med `useCapture` inställd på `true` och sedan en gång till inställd på `false`. Om du vill ta bort båda händelseavlyssnarna måste du anropa `removeEventListener()` två gånger, först med `useCapture` inställd på `true` och sedan en gång till inställd på `false`.

Skicka händelser

Metoden `dispatchEvent()` kan användas av erfarna programmerare för att skicka ett anpassat händelseobjekt till händelseflödet. Den enda parametern som godtas av den här metoden är en referens till ett händelseobjekt, som måste vara en instans av klassen Event eller en underklass till klassen Event. Efter att ha skickats ställs egenskapen `target` för händelseobjektet in på objektet där `dispatchEvent()` anropades.

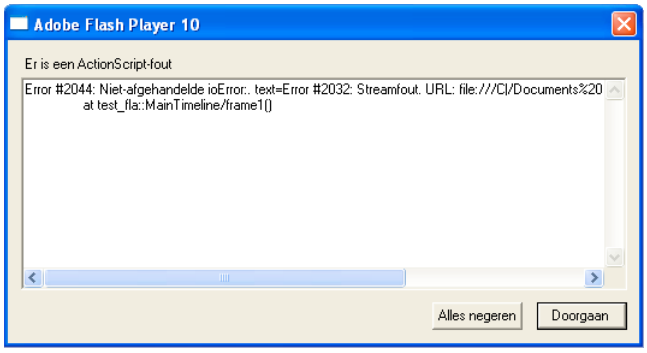
Söka efter befintliga händelseavlyssnare

De sista två metoderna i IEventDispatcher-gränssnittet erbjuder användbar information om förekomsten av händelseavlyssnare. Metoden `hasEventListener()` returnerar `true` om en händelseavlyssnare påträffas för en specifik händelsetyp i ett visst visningslisteobjekt. Metoden `willTrigger()` returnerar även `true` om en avlyssnare påträffas för ett visst visningslisteobjekt, men `willTrigger()` söker inte bara efter avlyssnare som finns i det visningsobjektet utan i alla visningsobjektets indirekta överordnade för alla faser av händelseflödet.

Felhändelser utan avlyssnare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Undantag, snarare än händelser, är den primära mekanismen för felhantering i ActionScript 3.0, men undantagshantering fungerar inte för asynkrona åtgärder som inläsning av filer. Om ett fel uppstår under en asynkron åtgärd skickar Flash Player och AIR ett felhändelseobjekt. Om du inte skapar en avlyssnare för den felhändelsen, kommer felsökningsversionerna av Flash Player och AIR att visa en dialogruta med information om felet. Felsökningsversionen av Flash Player skapar till exempel följande dialogruta som beskriver felet när programmet försöker att läsa in en fil från en ogiltig URL:



De flesta felhändelser är baserade på klassen ErrorEvent och som sådana har de en egenskap med namnet `text` som används för att lagra felmeddelandet som Flash Player eller AIR visar. De två undantagen är klasserna StatusEvent och NetStatusEvent. Båda dessa klasser har en `level`-egenskap (`StatusEvent.level` och `NetStatusEvent.info.level`). När värdet för `level`-egenskapen är `error`, betraktas dessa händelsetyper som felhändelser.

En felhändelse medför inte att körningen av en SWF-fil avbryts. Den visar sig bara i form av en dialogruta i felsökningsversioner av webbläsarplugin-program och fristående spelare, som ett meddelande i utdatapanelen i utvecklingsspelaren och som en post i loggfilen för Adobe Flash Builder. Den visas inte alls i de officiella versionerna av Flash Player eller AIR.

Exempel på händelsehantering: väckarklocka

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Exemplet Alarm Clock består av en klocka som gör att användaren kan ange vilken tid larmet ska utlösas och att ett meddelande då ska visas. Exemplet Alarm Clock bygger på SimpleClock-programmet från Alarm Clock. ”Arbeta med datum och tidpunkter” på sidan 1 illustrerar flera aspekter av att arbeta med händelser i ActionScript 3.0, inklusive följande:

- Lyssna efter och svara på en händelse
- Meddela avlyssnare om en händelse
- Skapa en anpassad händelsetyp

Information om hur du hämtar Flash Professional-programfilerna för det här exemplet finns i http://www.adobe.com/go/learn_programmingAS3samples_flash_se. Information om hur du hämtar Flex-programfilerna för det här exemplet finns i http://www.adobe.com/go/as3examples_se. Alarm Clock-programfilerna finns i mappen Samples/AlarmClock. Programmet innehåller följande filer:

Fil	Beskrivning
AlarmClockApp.mxml eller AlarmClockApp.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/clock/AlarmClock.as	En klass som utökar klassen SimpleClock genom att lägga till larmfunktionalitet.
com/example/programmingas3/clock/AlarmEvent.as	En anpassad händelseklass (en underklass till flash.events.Event) som fungerar som händelseobjektet för klassen AlarmClocks `alarm`-händelse.
com/example/programmingas3/clock/AnalogClockFace.as	Ritar en rund urtavla med tim-, minut- och sekundvisare baserat på tiden (enligt beskrivningen i SimpleClock-exemplet).
com/example/programmingas3/clock/SimpleClock.as	En klockgränssnittskomponent med enkel tidangivningsfunktionalitet (enligt beskrivningen i SimpleClock-exemplet).

Översikt över Alarm Clock

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den primära funktionen hos klockan i det här exemplet, inklusive att följa upp tiden och visa urtavlan, återanvänder SimpleClock-programkoden som beskrivs i ”Exempel på datum och tid: enkel analog klocka” på sidan 6. Klassen AlarmClock utökar klassen SimpleClock från det exemplet genom att lägga till larmfunktionalitet, inklusive inställning av väckningstid och meddelande om när klockan ”ringer”.

Att meddela när något händer är den uppgift som händelser är till för. Klassen AlarmClock visar Alarm-händelsen, som andra objekt kan lyssna efter för att kunna utföra önskade åtgärder. Klassen AlarmClock använder dessutom en instans av klassen Timer för att avgöra när larmet ska utlösas. Klassen Timer erbjuder liksom klassen AlarmClock en händelse för att meddela andra objekt (en AlarmClock-instans i det här fallet) när en viss tid har gått. Precis som med de flesta ActionScript-program utgör händelser en viktig del av funktionaliteten hos Alarm Clock-exempelprogrammet.

Utlösa larmet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som tidigare nämnts är den enda funktionaliteten som klassen AlarmClock erbjuder den som rör inställning och utlösning av larmet. Den inbyggda klassen Timer (flash.utils.Timer) erbjuder ett annat sätt för programmerare att definiera kod som ska köras efter en viss tid. Klassen AlarmClock använder en Timer-instans för att bestämma när larmet ska utlösas.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

Timer-instansen i klassen AlarmClock heter `alarmTimer`. `initClock()`-metoden, som utför nödvändiga inställningar för AlarmClock-instansen, gör två saker med `alarmTimer`-variabeln. Först skapas variabeln med parametrar som instruerar Timer-instansen att vänta 0 millisekunder och att bara utlösa timer-händelsen en gång. Efter att ha skapat `alarmTimer` anropar koden variabelns `addEventListener()`-metod för att ange att den vill lyssna på den variabelns `timer`-händelse. En Timer-instans fungerar så att den skickar sin `timer`-händelse efter att en viss tid har gått. Klassen AlarmClock behöver veta när `timer`-händelsen skickas för att kunna utlösa sitt eget larm. Genom att anropa `addEventListener()` registrerar AlarmClock-koden sig själv som avlyssnare med `alarmTimer`. De två parametrarna indikerar att klassen AlarmClock vill lyssna efter `timer`-händelsen (indikeras av konstanten `TimerEvent.TIMER`) och att klassen AlarmClocks `onAlarm()`-metod anropas som svar på händelsen när den inträffar.

För att verkligen kunna ställa in larmet anropas klassen AlarmClocks `setAlarm()`-metod som i följande:

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

Den här metoden gör flera saker, som att lagra larmmeddelandet och att skapa ett datumobjekt (`alarmTime`) som representerar den verkliga tidpunkten när larmet ska utlösas. Det som är mest relevant här är metodens sista rader, som innebär att `alarmTimer`-variabelns timer ställs in och aktiveras. Först anropas dess `reset()`-metod, vilket stoppar timern och återställer den om den redan är igång. Sedan subtraheras den aktuella tiden (som representeras av `now`-variabeln) från `alarmTime`-variabelns värde för att bestämma hur många millisekunder som måste gå innan larmet utlöses. Klassen Timer utlöser inte sin `timer`-händelse som en absolut tid, så det är denna relativa tidsskillnad som tilldelas `delay`-egenskapen för `alarmTimer`. Slutligen anropas `start()`-metoden för att starta timern.

När den angivna tiden har gått, skickar `alarmTimertimer`-händelsen. Eftersom klassen AlarmClock registrerade sin `onAlarm()`-metod som avlyssnare för den händelsen, och `timer`-händelsen inträffar, anropas `onAlarm()`.

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

En metod som registreras som händelseavlyssnare måste definieras med lämplig signatur (d.v.s. parameteruppsättningen och metodens returtyp). För att vara avlyssnare för klassen Timers `timer`-händelse måste en metod definiera en parameter vars datatyp är TimerEvent (flash.events.TimerEvent), som är en underklass till klassen Event. När Timer-instansen anropar sina händelseavlyssnare, överför den en TimerEvent-instans som händelseobjekt.

Meddela andra om larmet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen AlarmClock erbjuder liksom klassen Timer en händelse som gör att annan kod kan ta emot meddelanden när larmet utlöses. För att en klass ska använda det ramverk för händelsehantering som är inbyggt i ActionScript måste klassen implementera gränssnittet flash.events.IEventDispatcher. Detta görs normalt genom att utöka klassen flash.events.EventDispatcher, som erbjuder en standardimplementering av IEventDispatcher (eller genom att utöka en av underklasserna till EventDispatcher). Som tidigare har beskrivits utökar klassen AlarmClock klassen SimpleClock, som i sin tur (via en arvskedja) utökar klassen EventDispatcher. Allt detta innebär att klassen AlarmClock redan har inbyggd funktionalitet att erbjuda sina egna händelser.

Annan kod kan registreras så att den får meddelande om klassen AlarmClocks `alarm`-händelse genom att anropa `addEventListener()`-metoden som AlarmClock ärver från EventDispatcher. När en AlarmClock-instans är redo att meddela annan kod att dess `alarm`-händelse har utlösts, gör den det genom att anropa `dispatchEvent()`-metoden, som också ärvs från EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Dessa rader kod är tagna från klassen AlarmClocks `onAlarm()`-metod (som redan har visats fullständigt tidigare i den här dokumentationen). AlarmClock-instansens `dispatchEvent()`-metod anropas, som i sin tur meddelar alla registrerade avlyssnare att AlarmClock-instansens `alarm`-händelse har utlösts. Parametern som överförs till `dispatchEvent()` är det händelseobjekt som kommer att överföras till avlyssnarmetoderna. I det här fallet är detta en instans av klassen AlarmEvent, som är en Event-underklass särskilt framtagen för det här exemplet.

Erbjuda en anpassad larmhändelse

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla händelseavlyssnare får en händelseobjektparameter med information om den specifika händelse som utlöses. I många fall är händelseobjektet en instans av klassen Event. Ibland kan det dock vara användbart att erbjuda ytterligare information till händelseavlyssnare. Detta uppnås ofta genom att en ny klass definieras, en underklass till klassen Event, och därefter används en instans av den klassen som händelseobjekt. I det här exemplet används en AlarmEvent-instans som händelseobjekt när klassen AlarmClocks `alarm`-händelse skickas. Klassen AlarmEvent, som visas här, erbjuder ytterligare information om `alarm`-händelsen, nämligen larmmeddelandet:

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

Det bästa sättet att skapa en anpassad händelseobjektsklass på är att definiera en klass som utökar klassen Event, som i föregående exempel. För att komplettera den ärvda funktionaliteten definierar klassen AlarmEvent en `message`-egenskap som innehåller texten för larmmeddelandet som är kopplat till händelsen. Värdet för `message` anges som en parameter i AlarmEvent-konstruktorn. Klassen AlarmEvent definierar även konstanten `ALARM`, som kan användas för att referera till den specifika händelsen (`alarm`) vid anrop av klassen AlarmClocks `addEventListener()`-metod.

Förutom att lägga till anpassad funktionalitet måste varje Event-underklass åsidosätta den ärvda `clone()`-metoden som del av ramverket för ActionScript-händelsehantering. Event-underklasser kan även (valfritt) åsidosätta den ärvda `toString()`-metoden för att inkludera den anpassade händelsens egenskaper i värdet som returneras när `toString()`-metoden anropas.

```
    /**
     * Creates and returns a copy of the current instance.
     * @return A copy of the current instance.
     */
    public override function clone():Event
    {
        return new AlarmEvent(message);
    }

    /**
     * Returns a String containing all the properties of the current
     * instance.
     * @return A string representation of the current instance.
     */
    public override function toString():String
    {
        return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase",
"message");
    }
```

Den åsidosatta `clone()`-metoden måste returnera en ny instans av den anpassade Event-underklassen med alla anpassade egenskaper inställda så att de överensstämmer med den aktuella instansen. I den åsidosatta `toString()`-metoden används verktygsmetoden `formatToString()` (ärvd från Event) för att ange en sträng med namnet på den anpassade typen samt namnen och värdena för alla dess egenskaper.

Kapitel 9: Arbeta med programdomäner

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Syftet med klassen ApplicationDomain är att lagra en tabell med ActionScript 3.0-definitioner. All kod i en SWF-fil är definierad för att finnas i en programdomän. Programdomäner används för att partitionera klasser inom samma säkerhetsdomän. Detta gör det möjligt att använda flera definitioner av samma klass samt att underordnade klasser kan återanvända överordnade klassers definitioner.

Du kan använda programdomäner när du läser in en extern SWF-fil som har skrivits i ActionScript 3.0 med hjälp av API:et för klassen Loader. (Observera att du inte kan använda programdomäner för att läsa in en bild eller SWF-fil som har skrivits i ActionScript 1.0 eller ActionScript 2.0.) Alla ActionScript 3.0-definitioner som ingår i den inlästa klassen lagras i programdomänen. När du läser in SWF-filen kan du ange att filen ska inkluderas i samma programdomän som den för Loader-objektet. Det gör du genom att ställa in parametern `applicationDomain` för LoaderContext-objektet på `ApplicationDomain.currentDomain`. Genom att placera den inlästa SWF-filen i samma programdomän får du direktåtkomst till klasserna. Det är praktiskt om du läser in en SWF-fil som innehåller inbäddat medieinnehåll som du kan nå via det tillhörande klassnamnet eller om du vill nå den inlästa SWF-filens metoder.

I exemplet nedan har den har åtkomst till en separat Greeter.swf-fil som definierar en publik metod med namnet welcome():

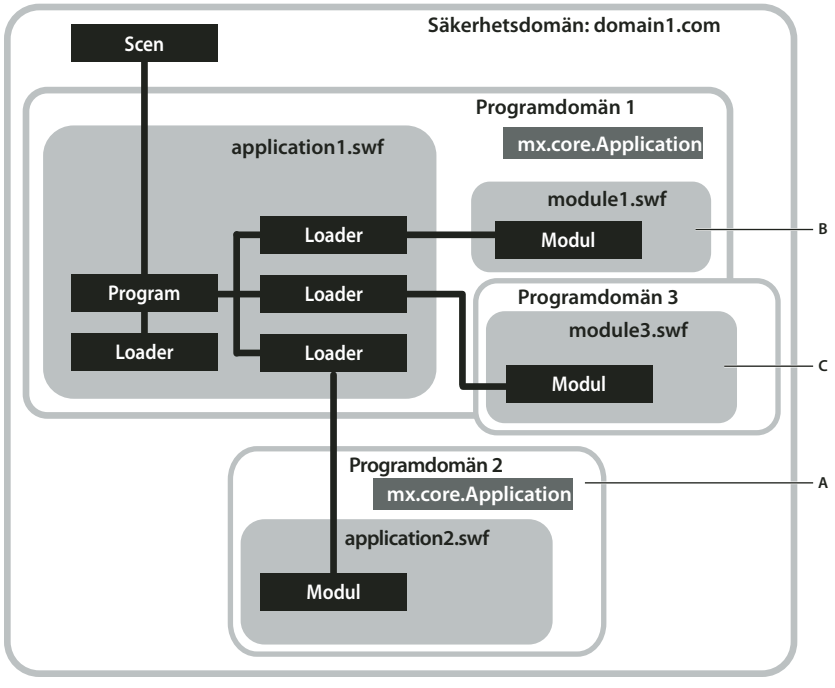
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    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
        {
            var myGreeter:Class = ApplicationDomain.currentDomain.getDefinition("Greeter") as
Class;
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Se även Exempel på klassen ApplicationDomain i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Du bör också tänka på följande när du arbetar med programdomäner:

- All kod i en SWF-fil är definierad för att finnas i en programdomän. Den *aktuella domänen* är den där huvudprogrammet körs. *Systemdomänen* innehåller alla programdomäner, inklusive den aktuella domänen, vilket betyder att den innehåller alla Flash Player-klasser.
- Alla programdomäner utom systemdomänen har en överordnad domän. Den överordnade domänen för huvudprogrammets programdomän är systemdomänen. Inlästa klasser definieras bara om de inte redan är definierade av sina överordnade klasser. Du kan inte åsidosätta en inläst klassdefinition med en nyare definition.

I nedanstående diagram visas ett program som läser in innehållet från olika SWF-filer i en enda domän, domain1.com. Beroende på vilket innehåll du läser in kan olika programdomäner användas. I den följande texten beskrivs logiken som används för att ställa in lämplig programdomän för varje SWF-fil i programmet.



A. *Användning A* ***B.*** *Användning B* ***C.*** *Användning C*

Huvudprogramfilen är application1.swf. Den innehåller Loader-objekt som läser in innehåll från andra SWF-filer. Enligt detta scenario är den aktuella domänen programdomän 1. Användning A, B och C illustrerar olika tekniker för att ställa in lämplig domän för varje SWF-fil i ett program.

Användning A Partitionera den underordnade SWF-filen genom att skapa en underordnad till systemdomänen. I diagrammet skapas programdomän 2 som underordnad till systemdomänen. Filen application2.swf läses in i programdomän 2, och dess klassdefinitioner partitioneras därmed från klasserna som är definierade i application1.swf.

Den här tekniken kan till exempel användas om du vill att ett äldre program dynamiskt ska läsa in en nyare version av samma program utan att någon konflikt uppstår. Även om samma klassnamn används uppstår ingen konflikt eftersom de är partitionerade i olika programdomäner.

I följande kod skapas en programdomän som är underordnad till systemdomänen, och en SWF-fil som använder den programdomänen börjar att läsas in:

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

Användning B: Lägga till nya klassdefinitioner till de aktuella klassdefinitionerna. Programdomänen för module1.swf är inställd på den aktuella domänen (programdomän 1). Det gör att du kan lägga till nya klassdefinitioner till programmets aktuella uppsättning med klassdefinitioner. Detta kan användas för ett delat körtidsbibliotek för huvudprogrammet. Den inlästa SWF-filen behandlas som ett fjärranslutet delat bibliotek (RSL). Använd den här tekniken om du vill läsa in RSL:er på förhand innan programmet startar.

I följande kod läses en SWF-fil in och programdomänen ställs in på den aktuella domänen:

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

Användning C: Använda de överordnade klassdefinitionerna genom att skapa en ny underordnad domän till den aktuella domänen. Programdomänen för module3.swf är underordnad den aktuella domänen, och den underordnade domänen använder den överordnade domänens versioner av samtliga klasser. Den här tekniken kan till exempel användas som en modul för RIA-program med flera skärmar som läses in som underordnad till huvudprogrammet, och använder huvudprogrammets typer. Om du kan se till att alla klasser alltid är uppdaterade och bakåtkompatibla, och att inläsningsprogrammet alltid är nyare än det som ska läsas in, kommer underordnade att använda de överordnades versioner. Genom att ha en ny programdomän kan du även ta bort alla klassdefinitioner för skräpsamling, om du kan se till att du inte fortsätter att ha referenser till den underordnade SWF-filen.

Den här tekniken gör att inlästa moduler kan dela inläsarens Singleton-objekt och statiska klassmedlemmar.

I följande kod skapas en ny underordnad domän till den aktuella domänen och en SWF-fil börjar läsas in med programdomänen:

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

Kapitel 10: Visningsprogrammering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

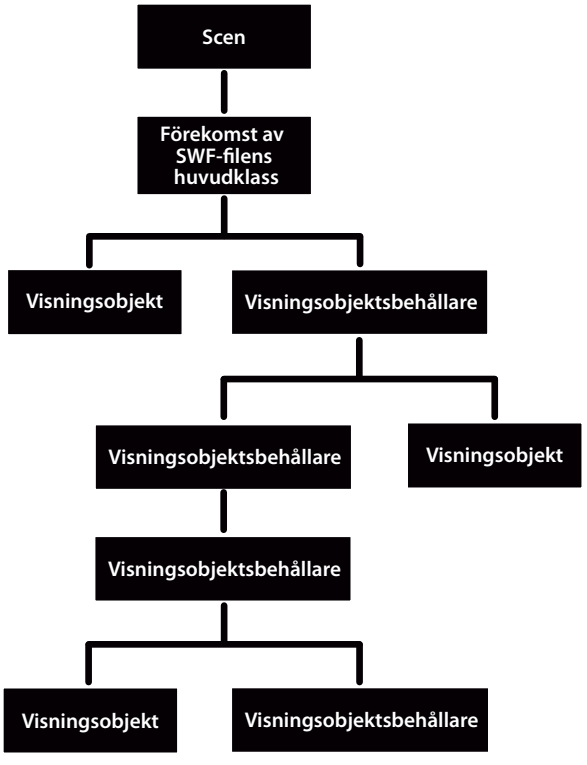
I Adobe® ActionScript® 3.0 programmerar du visuella element genom att arbeta med visningsobjekt på scenen. Du kan till exempel lägga till, flytta, ta bort och ordna visningsobjekt, använda filter och masker, rita vektor- och bitmappgrafik och utföra tredimensionella omformningar med ActionScripts API för visningsprogrammering. De primära klasserna som används för visningsprogrammering ingår som en del av flash.display-paketet.

Obs! *Adobe® AIR™ innehåller objektet HTMLLoader för återgivning och visning av HTML-innehåll. HTMLLoader återger visuella element i HTML DOM som ett enda visningsobjekt. Du kan inte komma åt de enskilda elementen i DOM direkt via visningshierarkin i ActionScript. I stället kommer du åt de här DOM-elementen genom att använda det separata DOM-API:t i HTMLLoader.*

Grunderna i visningsprogrammering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För alla program som skapats med ActionScript 3.0 finns en hierarki med visningsobjekt som kallas *visningslista*, vilket visas nedan. Visningslistan innehåller alla synliga element i programmet.



Så som visas i bilden hör visningselementen till en eller flera av följande grupper:

- Scenen

  Scenen är basbehållaren för alla visningsobjekt. I varje program finns ett scenobjekt, vilket innehåller alla visningsobjekt för skärmen. Scenen är behållaren på översta nivån och den visas högst upp i visningslistans hierarki:

  Varje SWF-fil har en tillhörande ActionScript-klass som heter *SWF-filens huvudklass*. När en SWF-fil öppnas i Flash Player eller Adobe AIR anropas konstruktorfunktionen för klassen och den instans som skapas (som alltid är en typ av visningsobjekt) läggs till som ett underordnat objekt till Stage-objektet. En SWF-fils huvudklass utökar alltid klassen Sprite (mer information finns i ”Fördelar med visningslistan” på sidan 156).

  Du får åtkomst till scenen via egenskapen `stage` i valfri DisplayObject-instans. Mer information finns i ”Inställning av scenegenskaper” på sidan 164.

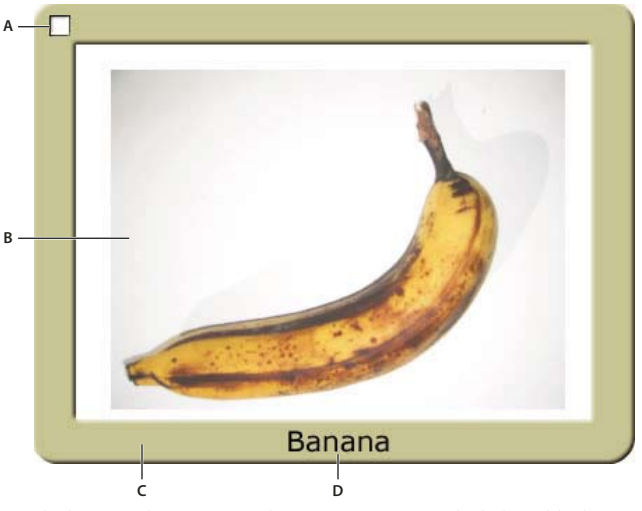
- Visningsobjekt

  I ActionScript 3.0 är alla element som visas på skärmen i ett program olika typer av *visningsobjekt*. I flash.display-paketet ingår klassen DisplayObject, som är en basklass utökad med ett antal andra klasser. Dessa olika klasser motsvarar olika typer av visningsobjekt, till exempel vektorformer, filmklipp och textfält. En översikt över dessa klasser finns i ”Fördelar med visningslistan” på sidan 156.

- Behållare för visningsobjekt

  Behållare för visningsobjekt är specifika typer av visningsobjekt som, utöver sin egen visuella representation, även kan innehålla underordnade objekt.

  Klassen DisplayObjectContainer är en underklass av DisplayObject-klassen. Ett DisplayObjectContainer-objekt kan innehålla flera visningsobjekt i sin *listamed underordnade objekt*. I följande exempel visas en typ av DisplayObjectContainer-objekt som heter Sprite och som innehåller olika visningsobjekt:



***A.** Objektet SimpleButton. Den här typen av visningsobjekt har olika lägen: upp, ned och över.* ***B.** Objektet Bitmap. I det här fallet lästes objektet Bitmap in från en extern JPEG-fil via objektet Loader.* ***C.** Objektet Shape. Tavelramen innehåller en rundad rektangulär form som ritats i ActionScript. I detta Shape-objekt används ett skuggfilter.* ***D.** Objektet TextField.*

DisplayObjectContainer-objekten kallas även *behållare för visningsobjekt* eller helt enkelt *behållare*. Som tidigare nämnts är scenen en behållare för visningsobjekt.

Trots att alla synliga visningsobjekt ärver från klassen DisplayObject är respektive typ en specifik underklass till klassen DisplayObject. Det finns till exempel en konstruktorfunktion för klassen Shape eller Video, men klassen DisplayObject saknar konstruktorfunktion.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kommer att stöta på när du programmerar ActionScript-grafik:

Alfa Färgvärdet som motsvarar hur pass genomskinlig en färg är (alltså mängden opacitet). En färg med till exempel ett alfakanalvärde på 60 % visar endast 60 % av hela sin styrka, vilket innebär 40 % genomskinlighet.

Bitmappsgrafik Grafik som i datorn har definierats som ett rutnät (rader och kolumner) med pixlar i olika färger. Vanlig bitmappsgrafik är till exempel digitala foton och liknande bilder.

Blandningsläge En specifikation över hur innehållet i två överlappande bilder ska samverka. En ogenomskinlig bild ovanpå en annan bild blockerar den underliggande bilden så att den inte syns alls. Men med olika blandningslägen blandas bildernas färger på olika sätt så att resultatet blir en kombination av de båda bilderna.

Visningslista Hierarkin för visningsobjekt som kommer att återges som synligt skärminnehåll av Flash Player och AIR. Scenen är roten i visningslistan och alla visningsobjekten som är kopplade till scenen eller något av underordnade objekt utgör visningslistan (även om objektet egentligen inte återges, till exempel om det ligger utanför gränserna för scenen).

Visningsobjekt Ett objekt som motsvarar någon typ av visuellt innehåll i Flash Player eller AIR. Endast visningsobjekt kan inkluderas i visningslistan och alla visningsobjektklasser är underklasser i klassen DisplayObject.

Behållare för visningsobjekt En specifik typ av visningsobjekt som kan innehålla underordnade visningsobjekt utöver att (vanligtvis) ha sin egen visuella representation.

SWF-filens huvudklass Klassen som definierar beteendet för det yttersta visningsobjektet i en SWF-fil, som per definition är klassen för själva SWF-filen. I en SWF-fil som skapats med Flash är till exempel dokumentklassen huvudklass. Den har en huvudtidslinje, som innehåller alla andra tidslinjer, och SWF-filens huvudklass är den klass där huvudtidslinjen är en instans.

Maskering En teknik som används för att dölja vissa delar av en bild (eller omvänt, endast tillåta visning av vissa delar av en bild). Delarna av maskbilden blir genomskinliga så att underliggande innehåll lyser igenom. Termen är relaterad till målarens maskeringstejp som används för att inte färgen ska hamna på vissa ytor.

Scen Den visuella behållaren som är bas eller bakgrund för allt visuellt innehåll i en SWF-fil.

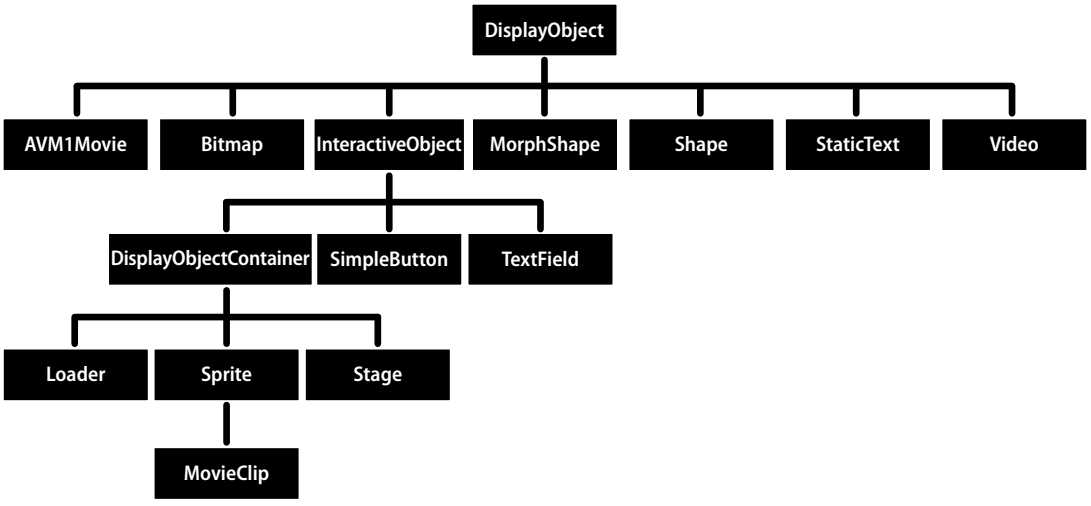
Omformning En justering av en visuell egenskap i en grafik, till exempel rotering av objektet, skaländring, skevning eller förvrängning av formen, eller ändring av färgen.

Vektorgrafik Grafik som definieras i datorn som linjer och former ritade med specifika egenskaper (till exempel tjocklek, längd, storlek, vinkel och position).

Huvudvisningsklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash.display-paketet i ActionScript 3.0 innehåller klasser för visuella objekt som kan visas i Flash Player eller AIR. I följande bild visas relationerna mellan dessa underklasser med huvudvisningsobjekt.



Bilden visar klassarv från visningsobjektklasser. Observera att några av klasserna, speciellt StaticText, TextField och Video, inte ingår i flash.display-paketet, men de ärver ändå från klassen DisplayObject.

Alla klasser som utökar klassen DisplayObject ärver motsvarande metoder och egenskaper. Mer information finns i ”Egenskaper och metoder i klassen DisplayObject” på sidan 158

Du kan instansiera objekt i följande klasser som ingår i flash.display-paketet:

- Bitmapp – använd klassen Bitmap för att definiera bitmappsobjekt som antingen är inlästa från externa filer eller återges via ActionScript. Du kan läsa in bitmappar från externa filer via klassen Loader. Du kan läsa in GIF-, JPG- eller PNG-filer. Du kan även skapa ett BitmapData-objekt med egna data och sedan skapa ett bitmappsobjekt där alla dessa data används. Metoderna i klassen BitmapData kan användas för att ändra bitmappar, oavsett om de är inlästa eller har skapats i ActionScript. Mer information finns i ”Inläsning av visningsobjekt” på sidan 198 och ”Arbeta med bitmappar” på sidan 243.
- Loader – använd klassen Loader när du ska läsa in externa objekt (antingen SWF-filer eller grafik). Mer information finns i ”Dynamisk inläsning av visningsinnehåll” på sidan 198.
- Shape – använd klassen Shape när du ska skapa vektorgrafik, till exempel rektanglar, linjer och cirklar. Mer information finns i ”Använda Drawing API (teckningsgränssnittet)” på sidan 222.
- SimpleButton – i ActionScript representerar detta objekt en knappsymbol som har skapats med Flash-utvecklingsverktyget. En SimpleButton-instans har fyra knapplägen: upp, ned, över och träfftest (området som svarar på mus- och tangentbordshändelser).
- Sprite – detta objekt kan innehålla egen grafik och underordnade visningsobjekt. (Klassen Sprite utökar klassen DisplayObjectContainer.) Mer information finns i ”Arbeta med behållare för visningsobjekt” på sidan 159 och ”Använda Drawing API (teckningsgränssnittet)” på sidan 222.

- MovieClip – Detta objekt är ActionScript-formen av en filmklippsymbol som skapats i Flash-utvecklingsverktyget. Ett filmklipp används ungefär som objektet Sprite, förutom att det även har en tidslinje. Mer information finns i ”Arbeta med filmklipp” på sidan 323.

Följande klasser, som inte ingår i flash.display-paketet, är underklasser i klassen DisplayObject:

- Klassen TextField, som ingår i flash.text-paketet, är ett visningsobjekt för textvisning och indata. Mer information finns i ”Grunderna för arbete med text” på sidan 371.
- Klassen TextLine, som ingår i paketet flash.text.engine, är det visningsobjekt som används för att visa textrader som skapats av Flash-textmotorn (FTE) och TLF (Text Layout Framework). Mer information finns i avsnitten ”Använda Flash-textmotorn” på sidan 397 och ”Använda Text Layout Framework” på sidan 426.
- Klassen Video, som ingår i flash.media-paketet, är visningsobjektet som används för visning av videofiler. Mer information finns i ”Arbeta med video” på sidan 474.

Följande klasser i flash.display-paketet utökar klassen DisplayObject, men du kan inte skapa instanser av dem. De fungerar i stället som överordnade klasser för andra visningsobjekt genom att kombinera funktioner i en enstaka klass.

- AVM1Movie – den här klassen används för att representera inlästa SWF-filer som har utvecklats i ActionScript 1.0 och 2.0.
- DisplayObjectContainer – klasserna Loader, Stage, Sprite och MovieClip utökar alla klassen DisplayObjectContainer. Mer information finns i ”Arbeta med behållare för visningsobjekt” på sidan 159.
- InteractiveObject – detta är basklassen för alla objekt som används vid samverkan med musen och tangentbordet. Objekten SimpleButton, TextField, Loader, Sprite, Stage och MovieClip är alla underklasser till klassen InteractiveObject. Mer information om hur du skapar samverkan med mus och tangentbord finns i ”Grunderna i användarinteraktion” på sidan 556.
- MorphShape – dessa objekt skapas när du skapar en interpoleringsform i Flash-utvecklingsverktyget. Du kan inte instansiera dem med ActionScript, men du har tillgång till dem via visningslistan.
- Stage – den här klassen utökar klassen DisplayObjectContainer. Det finns en sceninstans (Stage) per program och den finns överst i hierarkin för visningslistan. Du får åtkomst till scenen via egenskapen `stage` i valfri DisplayObject-instans. Mer information finns i ”Inställning av scenegenskaper” på sidan 164.

Klassen StaticText, i flash.text-paketet, utökar även klassen DisplayObject, men du kan inte skapa en instans av den i programkoden. Statiska textfält skapas endast i Flash.

Följande klasser är inte visningsobjekt eller behållare för visningsobjekt. De visas inte i visningslistan, men de visar grafik på scenen. De här klasserna ritar en rektangel, som kallas visningsruta och placeras i förhållande till scenen.

- StageVideo – Klassen StageVideo visar videoinnehåll, om möjligt med hjälp av maskinvaruacceleration. Den här klassen är tillgänglig från och med Flash Player 10.2. Mer information finns i ”Använda klassen StageVideo för maskinvaruaccelererad presentation” på sidan 512.
- StageWebView – Klassen StageWebView visar HTML-innehåll. Den här klassen är tillgänglig från och med AIR 2.5. Du Mer information finns i ”StageWebView-objekt” på sidan 1031.

Följande fl.display-klasser tillhandahåller funktioner som jämställer klasserna flash.display.Loader och LoaderInfo. Använd dessa klasser i stället för deras flash.display-motsvarighet om du arbetar med utveckling i Flash Professional-miljön (CS5.5 eller senare). I den miljön hjälper dig dessa klasser att lösa problem som har att göra med TLF med RSL-förinläsning. Mer information finns i ”Använda klasserna ProLoader och ProLoaderInfo” på sidan 202.

- fl.display.ProLoader – Motsvarar flash.display.Loader
- fl.display.ProLoaderInfo – Motsvarar flash.display.LoaderInfo

Fördelar med visningslistan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 finns det separata klasser för olika typer av visningsobjekt. I ActionScript 1.0 och 2.0 ingår många objekt av samma typ i en och samma klass: klassen MovieClip.

Den här åtskillnaden av klasser och den hierarkiska strukturen för visningslistor har följande fördelar:

- Effektivare återgivning och minskad minnesanvändning
- Förbättrad djuphantering
- Fullständig genomgång av visningslistan
- Visningsobjekt som inte är med i listan
- Enklare att skapa underklasser av visningsobjekt

Mer effektiv återgivning och mindre filstorlekar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 1.0 och 2.0 kan du rita former endast i MovieClip-objekt. I ActionScript 3.0 finns det enklare visningsobjektklasser som du kan rita former i. Eftersom dessa visningsobjektklasser i ActionScript 3.0 inte innehåller hela uppsättningen med metoder och egenskaper, som MovieClip-objekten gör, så används mindre minne och processorresurser.

Varje MovieClip-objekt innehåller till exempel egenskaper för tidslinjen för filmklippet, men inte Shape-objektet. Till egenskaperna som hanterar tidslinjen används mycket minnes- och processorresurser. Att använda Shape-objektet i ActionScript 3.0 ger bättre prestanda. Shape-objektet kräver mindre lagring än det mer komplexa MovieClip-objektet. Flash Player och AIR behöver inte hantera outnyttjade MovieClip-egenskaper, vilket ökar hastigheten och minskar minnesutrymmet som krävs för objektet.

Förbättrad djuphantering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 1.0 och 2.0 hanterades djupet via ett linjärt djuphanteringsschema och metoder som till exempel `getNextHighestDepth()`.

I ActionScript 3.0 ingår klassen DisplayObjectContainer som har mer praktiska metoder och egenskaper för djuphantering av visningsobjekt.

När du flyttar ett visningsobjekt i ActionScript 3.0 till en ny placering i listan med underordnade objekt för en DisplayObjectContainer-instans kommer övriga objekt i listan att placeras om automatiskt och tilldelas nya indexposter i behållaren för visningsobjekt.

I ActionScript 3.0 är det även möjligt att upptäcka alla underordnade objekt i valfri behållare för visningsobjekt. Varje DisplayObjectContainer-instans har en `numChildren`-egenskap som visar antalet underordnade objekt i behållaren för visningsobjekt. Och eftersom listan med underordnade objekt för en behållare för visningsobjekt alltid är en indexerad lista kan du granska alla objekt i listan från indexposition 0 till och med den sista indexpositionen (`numChildren - 1`). Detta var inte möjligt med metoderna och egenskaperna för ett MovieClip-objekt i ActionScript 1.0 och 2.0.

Det är enkelt att gå igenom visningslistan sekventiellt i ActionScript 3.0; det finns inga indexnummer som fattas i listan med underordnade objekt för behållaren för visningsobjekt. Att gå igenom visningslistan och hantera djupet av objekten är mycket enklare än vad som var möjligt i ActionScript 1.0 och 2.0. Ett filmklipp i ActionScript 1.0 och 2.0 kunde innehålla objekt med vissa mellanrum i djupordningen, vilket kunde försvåra genomgången av listan. I ActionScript 3.0 lagras varje lista med underordnade objekt för en behållare för visningsobjekt i cacheminnet som en array, vilket ger en mycket snabb sökning. Upprepad visning av alla underordnade objekt i en behållare för visningsobjekt fungerar också mycket snabbt.

I ActionScript 3.0 ges även åtkomst till underordnade objekt i en behållare för visningsobjekt genom användning av metoden `getChildByName()` i klassen DisplayObjectContainer.

Fullständig genomgång av visningslistan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 1.0 och 2.0 ges ingen åtkomst till vissa objekt, till exempel vektorformer, som ritades med Flash-utvecklingsverktyget. I ActionScript 3.0 har du åtkomst till alla objekt i visningslistan – både de som har skapats med ActionScript och de som har skapats i Flash-utvecklingsverktyget. Mer information finns i ”Gå igenom visningslistan” på sidan 163.

Visningsobjekt som inte är med i listan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med ActionScript 3.0 kan du skapa visningsobjekt som inte är synliga i visningslistan. Visningsobjekten är inte *med* i listan. Ett visningsobjekt läggs till i visningslistan endast om du anropar metoden `addChild()` eller `addChildAt()` för en DisplayObjectContainer-instans som redan finns med i visningslistan.

Du kan använda visningsobjekt som inte är med i listan när du ska sammankoppla komplexa visningsobjekt, till exempel dem som har flera behållare för visningsobjekt där flera visningsobjekt ingår. Genom att inte ta med visningsobjekt i listan kan du sammankoppla komplexa objekt utan använda bearbetningstid till att återge dessa visningsobjekt. Du kan sedan, vid behov, lägga till ett visningsobjekt som inte finns med i listan. Du kan även flytta ett underordnat objekt i en behållare för visningsobjekt till och från önskad position i listan.

Enklare att skapa underklasser av visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 1.0 och 2.0 behövde du ofta lägga till nya MovieClip-objekt till en SWF-fil för att skapa grundformer eller för att visa bitmappar. I ActionScript 3.0 innehåller klassen DisplayObjekt många inbyggda underklasser, till exempel Shape och Bitmap. Eftersom klasserna i ActionScript 3.0 är mer specialiserade för specifika typer av objekt är det enklare att skapa grundläggande underklasser i de inbyggda klasserna.

För att till exempel kunna rita en cirkel i ActionScript 2.0 kan du skapa klassen CustomCircle som omfattar klassen MovieClip när ett objekt från den egna klassen instansieras. Den klassen skulle däremot även innehålla ett antal egenskaper och metoder från klassen MovieClip (till exempel `totalFrames`) som inte används. I ActionScript 3.0 kan du däremot skapa klassen CustomCircle som omfattar objektet Shape och som inte innehåller orelaterade egenskaper och metoder från klassen MovieClip. I följande kod visas ett exempel med klassen CustomCircle:

```
import flash.display.*;

public 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);
    }
}
```

Arbeta med visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Nu när du känner till de grundläggande begreppen för scenen, visningsobjekt, behållare för visningsobjekt och visningslistan ger detta avsnitt ytterligare specifik information om hur du arbetar med visningsobjekt i ActionScript 3.0.

Egenskaper och metoder i klassen DisplayObject

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla visningsobjekt är underklasser till klassen DisplayObject och därför ärver de egenskaperna och metoderna i klassen DisplayObject. De ärvda egenskaperna är grundläggande egenskaper som används för alla visningsobjekt. Exempelvis har varje visningsobjekt egenskapen `x` och `y` som anger objektets position i sin behållare för visningsobjekt.

Du kan inte skapa en DisplayObject-instans med hjälp av klasskonstruktorn DisplayObject. Du måste skapa en annan typ av objekt (ett objekt som är en underklass till klassen DisplayObject), till exempel Sprite, för att instansiera ett objekt med `new`-operatorn. Om du dessutom vill skapa en egen visningsobjektklass måste du skapa en underklass till en av visningsobjektunderklasserna som har en användbar konstruktorfunktion (till exempel klassen Shape eller Sprite). Mer information finns i klassbeskrivningen för DisplayObject i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Lägga till visningsobjekt till visningslistan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du instansierar ett visningsobjekt kommer det inte att visas på skärmen (på scenen) förrän du lägger till visningsobjektsinstansen till en behållare för visningsobjekt som finns med i visningslistan. I följande kodexempel skulle inte TextField-objektet `myText` vara synligt ifall den sista raden i koden utelämnas. I den sista kodraden måste nyckelordet `this` hänvisa till en behållare för visningsobjekt som redan har lagts till i visningslistan.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

När du lägger till något visuellt element till scenen blir detta ett *underordnat element* till scenobjektet. Den första SWF-filen som läses in i ett program (till exempel den som du bäddar in i en HTML-sida) läggs automatiskt till som underordnad till scenen. Det kan vara ett objekt av valfri typ som omfattar klassen Sprite.

Alla visningsobjekt du skapar *utan* ActionScript – till exempel genom att lägga till en MXML-tagg i en Flex MXML-fil eller genom att placera ett objekt på scenen i Flash Professional – läggs till i visningslistan. Trots att du inte lägger till dessa visningsobjekt via ActionScript så har du åtkomst till dem därifrån. I följande kod visas hur du justerar bredden på objektet `button1` som las till i utvecklingsverktyget (inte via ActionScript):

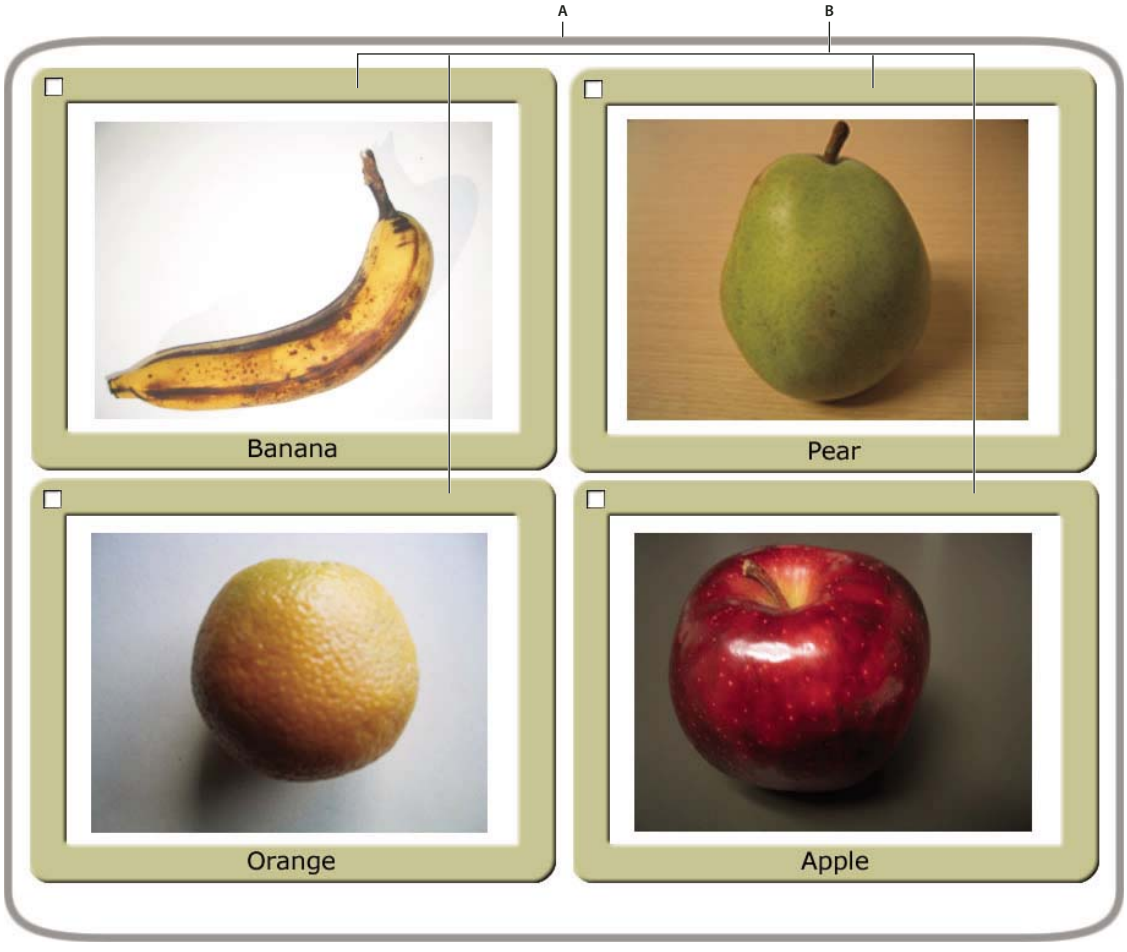
```
button1.width = 200;
```

Arbeta med behållare för visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om objektet DisplayObjectContainer tas bort från visningslistan, eller om det flyttas eller omformas på annat sätt, kommer även alla visningsobjekt i DisplayObjectContainer att tas bort, flyttas eller omformas.

En behållare för visningsobjekt är också en typ av visningsobjekt – det kan läggas till i en annan behållare för visningsobjekt. Följande bild illustrerar behållaren för visningsobjekt `pictureScreen` som innehåller en konturform och fyra andra behållare för visningsobjekt (av typen PictureFrame):



A. *En form som definierar kantlinjen för behållaren för visningsobjektet pictureScreen* ***B.*** *Fyra behållare för visningsobjekt som är underordnade objektet pictureScreen*

För att ett visningsobjekt ska kunna vara med i visningslistan måste du lägga till det till en behållare för visningsobjekt som är med i visningslistan. Detta gör du med hjälp av metoden `addChild()` eller `addChildAt()` i behållarobjektet. Utan den sista raden i följande kod skulle till exempel objektet `myTextField` inte visas.

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

I detta kodexempel pekar `this.root` till behållaren för visningsobjekt för MovieClip med motsvarande kod. I den faktiska koden kanske du anger en annan behållare.

Använd metoden `addChildAt()` för att lägga till det underordnade objektet till en specifik position i visningsobjektbehållarens lista med underordnade objekt. Dessa nollbaserade indexpositioner i listan med underordnade objekt relaterar till vilken ordning i lager (överst till underst) visningsobjekten har. Titta exempelvis på följande tre visningsobjekt. Alla objekten skapades från den egna klassen Ball.

Visningsobjektens ordning i behållaren kan justeras med hjälp av metoden `addChildAt()`. Titta på följande kod:

```
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);
```

Vid körning av den här koden placeras visningsobjekten enligt följande i DisplayObjectContainer-objektet `container`. Observera objektens ordning i lager.

Omplacera ett objekt till överst i visningslistan genom att lägga till det igen i listan. Om du till exempel efter föregående kod vill flytta `ball_A` till överst i stacken använder du följande kod:

```
container.addChild(ball_A);
```

Med den här koden tar du enkelt bort `ball_A` från positionen i visningslistan `container` och lägger till det överst i listan, vilket resulterar i att det läggs överst i stacken.

Du kan använda metoden `getChildAt()` för att kontrollera visningsobjektens ordningsföljd. Metoden `getChildAt()` returnerar underordnade objekt i en behållare baserat på det indexnummer du skickar. Med följande kod visas exempelvis namnen på visningsobjekten vid olika positioner i listan med underordnade objekt för DisplayObjectContainer-objektet `container`:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Om du tar bort ett visningsobjekt från överordnad behållares lista med underordnade kommer de högre elementen i listan att flyttas ner en position i det underordnade indexet. Som fortsättning på föregående kod visar koden nedan hur visningsobjektet från position 2 i `container` flyttas till position 1 när ett visningsobjekt som är lägre i listan med underordnade tas bort:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Metoderna `removeChild()` och `removeChildAt()` tar inte bort själva visningsobjektsinstansen. Den tas endast bort från listan med underordnade objekt för behållaren. Instansen kan fortfarande refereras av en annan variabel. (Använd operatorn `delete` för att helt ta bort ett objekt.)

Eftersom ett visningsobjekt endast har en överordnad behållare kan du lägga till en instans av ett visningsobjekt till endast en behållare för visningsobjekt. I följande kod ser du att visningsobjektet `tf1` kan finnas i endast en behållare (i detta fallet Sprite, som omfattar klassen DisplayObjectContainer):

```
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); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

Om du lägger till ett visningsobjekt som ingår i en visningsobjektbehållare till en annan behållare så tas den bort från den första behållarens lista med underordnade objekt.

Utöver ovan beskrivna metoder finns ytterligare metoder i klassen DisplayObjectContainer som du använder när du arbetar med underordnade visningsobjekt:

- `contains()`: Avgör om ett visningsobjekt är underordnat i DisplayObjectContainer.
- `getChildByName()`: Hämtar ett visningsobjekt per namn.
- `getChildIndex()`: Returnerar indexpositionen för ett visningsobjekt.
- `setChildIndex()`: Ändrar positionen för ett underordnat visningsobjekt.
- `removeChildren()`: Tar bort flera underordnade visningsobjekt.
- `swapChildren()`: Ändrar ordningen (överst till underst) för två visningsobjekt.

- `swapChildrenAt()`: Ändrar ordningen (överst till underst) för två visningsobjekt som anges via indexvärden.

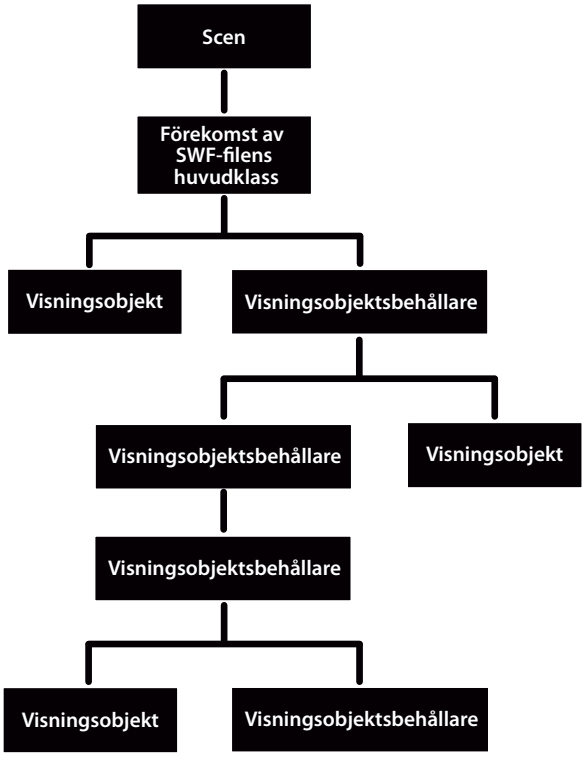
Du hittar mer information i respektive avsnitt av Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Kom ihåg att ett visningsobjekt *utanför visningslistan* inte ingår i en behållare för visningsobjekt som är underordnad scenen.

Gå igenom visningslistan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som du har sett är visningslistan en trädstruktur. Överst i trädet är scenen, som kan innehålla flera visningsobjekt. Dessa visningsobjekt som i sig själva är behållare för visningsobjekt kan innehålla andra visningsobjekt eller behållare för visningsobjekt.



Klassen DisplayObjectContainer innehåller egenskaper och metoder för genomgång av visningslistan med hjälp av visningsobjektbehållarnas listor med underordnade objekt. Titta på följande kod där två visningsobjekt läggs till, `title` och `pict`, till objektet `container` (som är Sprite och klassen Sprite omfattar klassen DisplayObjectContainer):

```
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);
```

Metoden `getChildAt()` returnerar underordnat objekt i visningslistan vid en specifik indexposition:

```
trace(container.getChildAt(0) is TextField); // true
```

Du kan även få åtkomst till underordnade objekt per namn. Alla visningsobjekt har en namnegenskap och om du inte tilldelar något namn tilldelas ett standardvärde i Flash Player eller AIR, till exempel `"instance1"`. Följande kod visar hur du använder metoden `getChildByName()` för att använda ett underordnat visningsobjekt med namnet `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

Användning av metoden `getChildByName()` kan innebära långsammare bearbetning än metoden `getChildAt()`.

Eftersom en behållare för visningsobjekt kan innehålla andra behållare som är underordnade objekt i visningslistan kan du gå igenom programmets hela visningslista i ett träd. I kodexemplet som visades tidigare ser du att när inläsningen för Loader-objektet `pict` är slutförd kommer `pict`objektet att innehålla ett underordnat visningsobjekt (bitmappen). När du ska använda denna bitmapp kan du skriva `pict.getChildAt(0)`. Du kan även skriva `container.getChildAt(0).getChildAt(0)` (eftersom `container.getChildAt(0) == pict`).

Följande funktion ger indragen `trace()`-utdata i visningslistan från en behållare för visningsobjekt:

```
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 + "")
        }
    }
}
```

Adobe Flex

Om du använder Flex bör du känna till att Flex definierar många visningsobjektklasser, och dessa klasser åsidosätter åtkomstmetoderna till visningslistan för klassen DisplayObjectContainer. Klassen Container för mx.core-paketet åsidosätter till exempel metoden `addChild()` och andra metoder för klassen DisplayObjectContainer (som klassen Container utökar). När det gäller metoden `addChild()` åsidosätter klassen metoden på så sätt att du inte kan lägga till alla typer av visningsobjekt i en Container-instans i Flex. Åsidosättningsmetoden kräver i detta fall att det underordnade objektet som du lägger till är av typen mx.core.UIComponent.

Inställning av scenegenskaper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Scenklassen åsidosätter de flesta egenskaper och metoder i klassen DisplayObject. Om du anropar en av dessa åsidosatta egenskaper eller metoder genereras ett undantag i Flash Player och AIR. Scenobjektet, exempelvis, har inte egenskaperna `x` eller `y` eftersom det har en fast position som programmets huvudbehållare. Egenskaperna `x` och `y` hänvisar till positionen hos ett visningsobjekt i relation till dess behållare. Eftersom scenen inte ingår i någon annan behållare för visningsobjekt används inte dessa egenskaper.

Obs! *En del egenskaper och metoder i scenklassen är bara tillgängliga för visningsobjekt som är i samma säkerhetssandlåda som den första inlästa SWF-filen. Mer information finns i* *"Scen, säkerhet" på sidan 1068.*

Kontrollera uppspelningens bildrutehastighet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskapen `framerate` i klassen Stage används för att ange bildrutehastigheten för alla inlästa SWF-filer i programmet. Du hittar mer information i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Kontrollera scenskalning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När storleken på den del av skärmen som visar Flash Player eller AIR ändras justeras sceninnehållet automatiskt. Egenskapen `scaleMode` för klassen Stage anger hur sceninnehållet justeras. Den här egenskapen kan anges med fyra olika värden, som är definierade som konstanter i klassen flash.display.StageScaleMode:

- `StageScaleMode.EXACT_FIT` skalar SWF-filen så att den fyller scenens nya storlek, utan hänsyn till det ursprungliga innehållets proportioner. Skalningsfaktorerna kanske inte är samma för bredden som för höjden, så innehållet kan se ihoptryckt eller utsträckt ut om scenens proportioner ändras.
- `StageScaleMode.SHOW_ALL` skalar SWF-filen så att den passar helt innanför scenens nya mått, utan att ändra innehållets proportioner. Det här skalningsläget visar allt innehåll, men kan ge upphov till s.k. letterbox-kanter, svarta fält som visas i kanterna när en bredbildsfilm visas på en standard-tv.
- `StageScaleMode.NO_BORDER` skalar SWF-filen så att den fyller scenens nya storlek, utan att ändra innehållets proportioner. Det här skalningsläget utnyttjar scenens visningsområde fullt ut, men kan orsaka beskärning.
- `StageScaleMode.NO_SCALE` skalar inte SWF-filen. Om scenens nya storlek är mindre beskärs innehållet, och om den är större blir de tillagda områdena tomma.

  I skalningsläget `StageScaleMode.NO_SCALE` kan egenskaperna `Width` och `Height` i klassen Stage användas för att avgöra de faktiska pixeldimensionerna för den nya scenstorleken. (I övriga skalningslägen återspeglar egenskaperna `stageWidth` och `stageHeight` alltid den ursprungliga bredden och höjden för SWF.) Dessutom, när `scaleMode` är angivet med `StageScaleMode.NO_SCALE` och SWF-filen har ändrat storlek, skickas scenklassens `resize`-händelse så att du kan göra de justeringar som behövs.

  Detta innebär, att om `scaleMode` är angivet med `StageScaleMode.NO_SCALE` får du större möjlighet att kontrollera hur skärminnehållet justeras till fönsterstorleken. I till exempel en SWF-fil som innehåller en video och ett kontrollfält kanske du vill att kontrollfältet ska ha bibehållen storlek om scenstorleken ändras; därmed behöver du ändra endast storleken på videofönstret för att justera enligt ändrad scenstorlek. Följande exempel visar detta:

```
// mainContent is a display object containing the main content;
// it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below mainContent) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = mainContent.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;
swfStage.addEventListener(Event.RESIZE, resizeDisplay);

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the main content area
    var newContentHeight:Number = swfHeight - controlBar.height;
    mainContent.height = newContentHeight;
    mainContent.scaleX = mainContent.scaleY;

    // Reposition the control bar.
    controlBar.y = newContentHeight;
}
```

Ställa in scenskalningslägen för AIR-fönster

Scenens `scaleMode`-egenskap avgör hur underordnade visningsobjekt skalas och klipps när storleken på ett fönster ändras. Endast `noScale`-läget bör användas i AIR. I det här läget skalas inte scenen. I stället ändras scenens storlek inom fönstrets gränser. Objekt kan klippas bort om storleken på fönstret minskas.

Scenskalningslägen är utformade för webbläsare och liknande miljöer, där du inte alltid har kontroll över scenens storlek eller proportioner. Med hjälp av dessa lägen kan du välja den bästa kompromissen för situationer när scenen inte stämmer överens med den ideala storleken eller proportionerna för programmet. I AIR har du alltid kontroll över scenen, och i de flesta fall kan du få ett bättre resultat genom att göra om layouten för innehållet eller justera fönsterstorleken, i stället för att aktivera skalning av scenen.

I webbläsaren och det ursprungliga AIR-fönstret läses förhållandet mellan fönsterstorleken och den ursprungliga skalfaktorn in från den inlästa SWF-filen. Om du däremot skapar ett NativeWindow-objekt väljer AIR ett arbiträrt förhållande mellan fönsterstorleken och skalfaktorn 72:1. Det betyder att om fönstret är 72 x 72 pixlar ritas en rektangel med storleken 10 x 10 som har lagts till i fönstret med den korrekta storleken 10 x 10 pixlar. Om fönstret är 144 x 144 pixlar skalas en rektangel med storleken 10 x 10 pixlar till 20 x 20 pixlar. Om du fortfarande vill använda `scaleMode` i stället för `noScale` för en fönsterscen, kan du kompensera genom att ställa in skalfaktorn för visningsobjekten i fönstret till skalan 72 pixlar till aktuell bredd och höjd för scenen. Följande kod beräknar till exempel den skalfaktor som krävs för ett visningsobjekt med namnet `client`:

```
if(newWindow.stage.scaleMode != StageScaleMode.NO_SCALE){
client.scaleX = 72/newWindow.stage.stageWidth;
client.scaleY = 72/newWindow.stage.stageHeight;
}
```

Obs! *I Flex- och HTML-fönster ställs scenen automatiskt in från* `scaleMode` *till* `noScale`*. Om du ändrar* `scaleMode` *stör det den automatiska layoutfunktionen som används i den här typen av fönster.*

Arbeta i helskärmsläge

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med helskärmsläge kan du ställa in en filmscen så att den fyller hela bildskärmen, utan några behållarkanter eller menyer. Scenklassens egenskap `displayState` används till att växla mellan helskärmsläge och vanligt läge för en SWF-fil. Egenskapen `displayState` kan anges med något av värdena som finns definierade med konstanterna i klassen flash.display.StageDisplayState. Om du vill växla till helskärmsläge tilldelar du egenskapen `displayState` värdet `StageDisplayState.FULL_SCREEN`:

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

Om du vill aktivera interaktivt helskärmsläge (nytt i Flash Player 11.3) anger du egenskapen `displayState` som `StageDisplayState.FULL_SCREEN_INTERACTIVE`:

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

Helskärmsläge i Flash Player kan endast initieras via en musklickning (även högerklick) eller tangenttryckning i ActionScript. AIR-innehåll som körs i säkerhetssandlådan application kräver inte att helskärmsläget används som svar på en användargest.

Om du vill stänga helskärmsläget tilldelar du egenskapen `displayState` värdet `StageDisplayState.NORMAL`.

```
stage.displayState = StageDisplayState.NORMAL;
```

Dessutom kan användaren välja att avbryta helskärmsläget genom att växla fokus till ett annat fönster eller genom att använda någon av flera tangentkombinationer: Esc-tangenten (alla plattformar), Ctrl-W (Windows), Kommando-W (Mac) eller Alt-F4 (Windows).

Aktivera helskärmsläge i Flash Player

Om du vill aktivera helskärmsläge för en SWF-fil som är inbäddad på en HTML-sida måste HTML-koden som bäddar in Flash Player innehålla en `param`-tagg och ett `embed`-attribut med namnet `allowFullScreen` och värdet `true` enligt följande:

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowFullScreen="true" />
</object>
```

Välj Arkiv -> Publiceringsinställningar i utvecklingsverktyget i Flash och välj sedan mallen Endast Flash - Tillåt helskärm på fliken HTML i dialogrutan Publiceringsinställningar.

I Flex ser du till att HTML-mallen omfattar `<object>`- och `<embed>`-taggarna som stöder helskärm.

Om du använder JavaScript i en webbsida för att generera SWF-inbäddande taggar måste du ändra JavaScript-koden och lägga till taggen och attributet `allowFullScreen param`. Om HTML-sidan till exempel använder funktionen `AC_FL_RunContent()` (som används på HTML-sidor som genererats av Flash Professional och Flash Builder) bör du lägga till parametern `allowFullScreen` i det funktionsanropet enligt följande:

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Detta gäller inte för SWF-filer som körs i den fristående Flash Player.

Obs! *Om du ställer in fönsterläget (wmode i HTML) på ogenomskinlig fönsterlös (opaque) eller genomskinlig fönsterlös (transparent) blir helskärmsfönstret ogenomskinligt*

Det finns också säkerhetsrelaterade restriktioner som begränsar användningen av helskärmsläge med Flash Player i en webbläsare. Dessa begränsningar beskrivs i avsnittet ”Säkerhet” på sidan 1047.

Aktivera interaktivt helskärmsläge i Flash Player 11.3 och senare

Flash Player 11.3 och senare versioner har stöd för interaktivt helskärmsläge, vilket ger fullständigt stöd för alla tangentbordstangenter (utom **Esc**, som avslutar det interaktiva helskärmsläget). Helskärmsläge är användbart för spel (till exempel för att chatta i ett fleranvändarspel eller för WASD-tangentbordskontroller i ett FPS-spel (First-Person Shooter)).

Om du vill aktivera interaktivt helskärmsläge för en SWF-fil som bäddats in på en HTML-sida måste HTML-koden för att bädda in Flash Player innehålla en `param`-tagg och ett `embed`-attribut med namnet `allowFullScreenInteractive` och värdet `true`, enligt nedanstående:

```
<object>
    ...
    <param name="allowFullScreenInteractive" value="true" />
    <embed ... allowFullScreenInteractive="true" />
</object>
```

Välj Arkiv -> Publiceringsinställningar i utvecklingsverktyget i Flash och välj sedan mallen Endast Flash - Tillåt helskärm på fliken HTML i dialogrutan Publiceringsinställningar.

I Flash Builder och Flex kontrollerar du att HTML-mallen innehåller taggarna `<object>` och `<embed>`, som har stöd för interaktivt helskärmsläge.

Om du använder JavaScript på en webbsida för att generera taggarna för SWF-inbäddning måste du ändra JavaScript-koden och lägga till taggen `allowFullScreenInteractive param` och attributet. Om HTML-sidan till exempel använder funktionen `AC_FL_RunContent()` (som används på HTML-sidor som genererats av Flash Professional och Flash Builder) bör du lägga till parametern `allowFullScreenInteractive` i det funktionsanropet enligt följande:

```
AC_FL_RunContent(
    ...
    'allowFullScreenInteractive','true',
    ...
    ); //end AC code
```

Detta gäller inte för SWF-filer som körs i den fristående Flash Player-versionen.

Scenstorlek och skalning i helskärmsläge

Egenskaperna `Stage.fullScreenHeight` och `Stage.fullScreenWidth` returnerar höjd och bredd på bildskärmen som används vid helskärmsstorleken, om läget aktiveras omedelbart. Dessa värden kan bli fel om användaren har möjlighet att flytta webbläsaren från en bildskärm till en annan efter det att värdena har hämtats men innan helskärmsläget aktiveras. Om du hämtar värdena i samma händelsehanterare där egenskapen `Stage.displayState` tilldelats värdet `StageDisplayState.FULL_SCREEN`, stämmer värdena. Om en användare har flera bildskärmar expanderas SWF-innehållet så att bara en bildskärm fylls. I Flash Player och AIR används en mätmetod för att avgöra vilken bildskärm som innehåller den största andelen av SWF-innehåll och använda denna vid helskärmsläge. Egenskaperna fullScreenHeight och fullScreenWidth återspeglar endast storleken på den skärm som används i helskärmsläge. Du hittar mer information i avsnitten om `Stage.fullScreenHeight` och `Stage.fullScreenWidth` i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Scenskalning vid helskärmsläge är samma som vid normalläge; skalningen kontrolleras via scenklassens egenskap `scaleMode`. Om egenskapen `scaleMode` är angiven med `StageScaleMode.NO_SCALE` ändras scenegenskaperna `stageWidth` och `stageHeight` och återspeglar storleken på skärmarean som innehåller SWF-filen (i detta fall hela skärmen), om den visas i webbläsaren med HTML-parametern för den här inställningen.

Du kan använda scenklassens händelse `fullScreen` för att ta reda på och reagera på huruvida helskärmsläget används eller inte. Du kanske vill omplacera, lägga till eller ta bort objekt från skärmen när helskärmsläge används, som i detta exempel:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

Som du ser i koden är händelseobjektet för händelsen `fullScreen` en instans av klassen flash.events.FullScreenEvent, som innehåller egenskapen `fullScreen` som anger om helskärmsläget används (`true`) eller inte (`false`).

Stöd för tangentbord i helskärmsläge

Alla tangentbordsrelaterade ActionScript, till exempel tangentbordshändelser och textinmatning i TextField-instanser, inaktiveras i helskärmsläge när Flash Player körs i en webbläsare. Undantagen (de tangenter som aktiveras) är:

- valda tangenter som inte skrivs ut, t.ex. piltangenterna, blankstegstangenten och tabbtangenten
- kortkommandon som inaktiverar helskärmsläget: Esc (Windows och Mac), Ctrl-W (Windows), Command-W (Mac) och Alt-F4.

Dessa begränsningar gäller inte för SWF-innehåll som körs i den fristående Flash Player eller i AIR. AIR stöder ett interaktivt helskärmsläge som tillåter indata från tangentbordet.

Stöd för mus i helskärmsläge

Som standard fungerar mushändelser i helskärmsläge på samma sätt som utan helskärmsläge. I helskärmsläge kan du dock ange egenskapen `Stage.mouseLock` för att aktivera låsning av musen, om du vill. Om du låser musen inaktiveras markören och obegränsade musrörelser aktiveras.

Obs! *Du kan bara aktivera låsning av musen i helskärmsläge för skrivbordsprogram. Om du anger det på program som inte är i helskärmsläge, eller program på mobila enheter, genereras ett undantagsfel.*

Låsning av musen inaktiveras automatiskt, och musmarkören blir synlig igen när:

- Användaren avslutar helskärmsläget med Esc-tangenten (alla plattformar), Ctrl-W (Windows), Kommando-W (Mac) eller Alt-F4 (Windows).
- Programfönstret förlorar fokus.
- Eventuella inställningsgränssnitt är synliga, inklusive alla sekretessdialogrutor.
- En systemspecifik dialogruta visas, t.ex. en dialogruta för filöverföring.

Händelser som är kopplade till musrörelser, som `mouseMove`-händelsen, använder klassen MouseEvent för att representera händelseobjektet. När låsning av musen är inaktiverat använder du egenskaperna `MouseEvent.localX` och `MouseEvent.localY` för att fastställa var musen finns. När låsning av musen är aktiverat använder du egenskaperna `MouseEvent.movementX` och `MouseEvent.movementY` för att fastställa var musen finns. Egenskaperna `movementX` och `movementY` innehåller förändringar av musens position sedan den senaste händelsen, i stället för absoluta koordinater för musens position.

Maskinvaruskalning i helskärmsläge

Med scenklassens `fullScreenSourceRect`-egenskap kan du ställa in Flash Player eller AIR så att ett visst område på scenen skalas efter helskärmsläget. Om det går skalar Flash Player och AIR maskinvaran med hjälp av grafik- och bildskärmskortet på användarens dator. Innehållet visas vanligtvis snabbare med maskinvaruskalning än med programvaruskalning.

För att kunna dra nytta av maskinvaruskalning ställer du in hela scenen eller en del av scenen på helskärmsläge. I följande ActionScript 3.0-kod ställs hela scenen in på helskärmsläge:

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

När den här egenskapen är inställd på en giltig rektangel och egenskapen `displayState` är inställd på helskärmsläge skalar Flash Player och AIR det angivna området. Scenens faktiska storlek i pixlar i ActionScript ändras inte. Flash Player och AIR tillämpar en lägsta gräns för rektangelns storlek för att standardmeddelandet ”Tryck på Esc för att avsluta helskärmsläget” ska få plats. Gränsen är vanligen ungefär 260 gånger 30 pixlar, men den kan variera beroende på plattform och Flash Player-version.

Egenskapen `fullScreenSourceRect` kan bara ställas in när Flash Player inte är i helskärmsläge. Om du använder den här egenskapen är det viktigt att du först definierar egenskapen och sedan tilldelar egenskapen `displayState` värdet för helskärmsläge.

Aktivera skalning genom att ställa in egenskapen `fullScreenSourceRect` på ett rektangelobjekt.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

Om du vill inaktivera skalning tilldelar du egenskapen `fullScreenSourceRect` värdet `null`.

```
stage.fullScreenSourceRect = null;
```

Om du vill utnyttja alla funktioner för maskinvaruacceleration med Flash Player aktiverar du dem i dialogrutan Flash Player-inställningar. Du öppnar dialogrutan genom att högerklicka (Windows) eller Control-klicka (Mac) i Flash Player-innehållet i webbläsaren. Markera fliken Visning, som är den första fliken, och klicka i kryssrutan: Aktivera maskinvaruacceleration.

Direkt och GPU-sammansatt fönsterläge

Flash Player 10 innehåller två nya fönsterlägen, direkt och GPU-sammansättning, som du kan aktivera via publiceringsinställningarna i utvecklingsverktyget i Flash. Dessa lägen stöds inte i AIR. Om du vill använda dessa lägen måste du aktivera maskinvaruacceleration för Flash Player.

Direkt läge använder den snabbaste, mest direkta, vägen för att skicka grafiken till skärmen, vilket är en fördel vid videouppspelning.

GPU-sammansättning använder grafikprocessorn på bildskärmskortet för att accelerera sammansättningen. Videosammansättning är en process för att placera flera bilder i lager och skapa en enda videobild. När sammansättningen accelereras med GPU kan prestanda i YUV-konvertering, färgkorrigering, rotation eller skalning samt blandning förbättras. YUV-konvertering är den färgkonvertering av sammansatta analoga signaler – som används för överföring – till RGB-färgmodellen (röd, grön, blå) som videokameror och bildskärmar använder. Om du använder GPU för att accelerera sammansättning minskar behovet av minne och datorkraft i processorn. Det blir också en jämnare uppspelning av video med standarddefinition.

Var försiktig när du implementerar dessa fönsterlägen. Att använda GPU-sammansättning kan kräva mycket minne och processorresurser. Om vissa åtgärder (till exempel blandningslägen, filtrering, urklipp eller maskering) inte kan utföras i GPU, utförs de av programvaran. Adobe rekommenderar att du begränsar dig till en SWF-fil per HTML-sida när du använder dessa lägen och du bör inte aktivera dessa lägen för banderoller. Funktionen Flash-alternativet Testa filmen har inte maskinvaruacceleration, men du kan använda den via alternativet Förhandsgranska publicering.

Om du ställer in en bildhastighet i SWF-filen som är högre än 60, är den maximala uppdateringsfrekvensen oanvändbar. Om du ställer in bildhastigheten på 50 till och med 55 kan bildrutor uteslutas, vilket då och då kan ske av olika anledningar.

Om du använder direkt läge krävs Microsoft® DirectX 9 med 128 MB VRAM i Windows och OpenGL för Apple Macintosh, Mac OS X v10.2 eller senare. För GPU-sammansättning krävs Microsoft DirectX 9 och Pixel Shader 2.0-stöd i Windows med 128 MB VRAM. Med Mac OS X och Linux krävs OpenGL 1.5 och flera OpenGL-tillägg (bildrutebuffertobjekt, flera texturer, skuggningsobjekt, skuggningsspråk och fragmentskuggning) för GPU-sammansättning.

Du kan aktivera `direkt`- och `GPU`-accelerationsläget per SWF-fil via Flash-dialogrutan Publiceringsinställningar, med hjälp av menyn Maskinvaruacceleration på fliken Flash. Om du väljer Ingen återgår fönsterläget till `standard`, `genomskinlig` eller `ogenomskinlig`, beroende på vad som anges i inställningen för fönsterläge på fliken HTML.

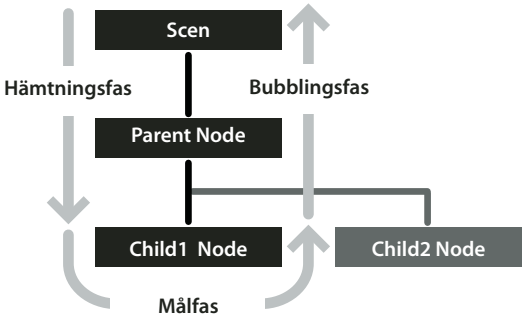
Hantera händelser för visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen DisplayObject ärver från klassen EventDispatcher. Detta innebär att alla visningsobjekt kan delta helt i händelsemodellen (beskrivs i ”Hantera händelser” på sidan 125). Alla visningsobjekt kan använda respektive `addEventListener()`-metod, som ärvts från klassen EventDispatcher, för att avlyssna en specifik händelse, men endast om avlyssnande objekt är en del av händelseflödet för den händelsen.

När Flash Player och AIR skickar ett händelseobjekt gör detta en rundtur från scenen till visningsobjektet där händelsen inträffade. Om en användare till exempel klickar på ett visningsobjekt med namnet `child1` skickas ett händelseobjekt från scenen via visningslistans hierarki ner till visningsobjektet `child1`.

Händelseflödet är begreppsmässigt uppdelat i tre faser enligt följande diagram:



Mer information finns i ”Hantera händelser” på sidan 125.

När du arbetar med visningsobjekthändelser är det viktigt att tänka på hur händelseavlyssnare kan påverka ifall visningsobjekt automatiskt tas bort från minnet (skräpsamling) när de tas bort från visningslistan. Om ett visningsobjekt har objekt som fungerar som avlyssnare till händelser kommer det visningsobjektet inte att tas bort från minnet även om det tas bort från visningslistan, eftersom det fortfarande finns referenser till dessa avlyssningsobjekt. Mer information finns i ”Hantera händelseavlyssnare” på sidan 138.

Välja underklassen DisplayObject

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns flera alternativ att välja mellan, men ett av de viktigaste beslut du måste ta när du ska arbeta med visningsobjekt är vilket visningsobjekt som ska användas i vilket syfte. Här är några riktlinjer som kan underlätta ditt beslut. Samma rekommendationer gäller oavsett om du behöver en instans av en klass eller du väljer en basklass för en klass du tänker skapa:

- Om du inte behöver ett objekt som kan vara en behållare för andra visningsobjekt (d.v.s. du behöver endast ett objekt som fungerar som fristående bildskärmselement) så väljer du en av dessa DisplayObject- eller InteractiveObject-underklasser, beroende på vad den ska användas till:
  - Bitmap för att visa en bitmappsbild.
  - TextField för att lägga till text.
  - Video för att visa video.
  - Shape för en arbetsyta för att rita innehåll på skärmen. Om du vill skapa en instans för att rita former på skärmen, och det inte blir en behållare för andra visningsobjekt, kommer du att få avsevärt bättre prestanda med Shape i stället för Sprite eller MovieClip.
  - MorphShape, StaticText eller SimpleButton för objekt som har skapats med Flash-utvecklingsverktyget. (Du kan inte skapa instanser av dessa klasser genom programmering, men du kan skapa variabler med dessa datatyper för att referera till objekt skapade med hjälp av Flash-utvecklingsverktyget.)
- Om du behöver en variabel som refererar till huvudscenen använder du klassen Stage som datatyp.
- Om du behöver en behållare för inläsning av en extern SWF-fil eller bildfil så använder du en Loader-instans. Det inlästa innehållet läggs till i visningslistan som underordnat objekt för Loader-instansen. Dess datatyp beror på vad det inlästa innehållet är, enligt följande:
  - En inläst bild blir en Bitmap-instans.

  - En inläst SWF-fil skriven i ActionScript 3.0 blir en Sprite- eller MovieClip-instans (eller en instans av en underklass till någon av dessa klasser, enligt innehålls-creator).
  - En inläst SWF-fil skriven i ActionScript 1.0 eller ActionScript 2.0 blir en AVM1Movie-instans.
- Om du behöver ett objekt som ska användas som en behållare för andra visningsobjekt (oavsett om du även kommer att rita till visningsobjektet med ActionScript) väljer du en av underklasserna DisplayObjectContainer:
  - Sprite om objektet ska skapas med enbart ActionScript eller som basklass för ett anpassat visningsobjekt som ska skapas och ändras endast med ActionScript.
  - MovieClip om du ska skapa en variabel som refererar till en filmklippssymbol som skapats i utvecklingsverktyget.
- Om du ska skapa en klass som ska associeras med en filmklippssymbol i Flash-biblioteket väljer du en av dessa DisplayObjectContainer-underklasser som klassens basklass:
  - MovieClip om associerad filmklippssymbol har innehåll på fler än en bildruta
  - Sprite om associerad filmklippssymbol har innehåll på endast den första bildrutan

Ändra visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Oavsett vilket visningsobjekt du tänker använda finns det ett flertal ändringar som alla visningsobjekt har gemensamt som element för visning på bildskärmen. Alla kan till exempel placeras på bildskärmen, flyttas framåt eller bakåt i staplingsordningen för visningsobjekt, skaländras, roteras o.s.v. Eftersom alla visningsobjekt ärver denna funktion från sin gemensamma basklass (DisplayObject) är funktionen densamma, oavsett om du ändrar en TextField-, Video- eller Shape-instans eller något annat visningsobjekt. I följande avsnitt beskrivs flera av dessa ändringsmöjligheter som visningsobjekten har gemensamt.

Ändra position

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den vanligaste ändringen av ett visningsobjekt är positionen på bildskärmen. För att ange ett visningsobjekts position ändrar du objektets `x` och `y`-egenskaper.

```
myShape.x = 17;
myShape.y = 212;
```

Positionssystemet för visningsobjekt behandlar scenen som ett kartesiskt koordinatsystem (ett rutnätsystem med vågrät x-axel och lodrät y-axel). Koordinatsystemets origo (koordinaten 0,0 där x- och y-axeln möts) ligger i det översta vänstra hörnet på scenen. Därifrån är x-värden positiva åt höger och negativa åt vänster. Däremot (till skillnad från vanliga grafiksystem) är y-värden positiva nedåt och negativa uppåt. I till exempel föregående kodrader flyttas objektet `myShape` till x-koordinaten 17 (17 pixlar till höger om origo) och y-koordinaten 212 (212 pixlar nedanför origo).

När ett visningsobjekt skapas med ActionScript får `x`- och `y`-egenskaperna standardvärdet 0 och objektet placeras överst i vänstra hörnet i överordnat innehåll.

Ändra position i förhållande till scenen

Det är viktigt att komma ihåg att `x`- och `y`-egenskaperna alltid refererar till visningsobjektets position i förhållande till 0,0-koordinaten för överordnat visningsobjekt. För en Shape-instans (till exempel en cirkel) som ingår i en Sprite-instans där Shape-objektets `x`- och `y`-egenskaper anges som 0 placeras cirkeln överst i vänstra hörnet i Sprite-instansen, vilket inte behöver vara överst i vänstra hörnet på scenen. För att placera ett objekt i förhållande till de globala scenkoordinaterna kan du använda metoden `globalToLocal()` för valfritt visningsobjekt så att koordinaterna konverteras från globala (scen) till lokala (behållare för visningsobjekt) enligt följande:

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
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();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

Du kan på liknande sätt använda klassen DisplayObjects metod `localToGlobal()` för att konvertera lokala koordinater till scenkoordinater.

Flytta visningsobjekt med musen

Du kan göra det möjligt för en användare att flytta visningsobjekt med musen på två olika sätt i ActionScript. I båda fallen används två mushändelser: när musknappen trycks ner följer objektet med musmarkörens rörelse och när knappen släpps stannar objektet.

Obs! *Flash Player 11.3 och senare, AIR 3.3 och senare: Du kan även använda händelsen MouseEvent.RELEASE_OUTSIDE för att täcka in de fall då en användare släpper upp musknapen utanför gränserna för aktuellt Sprite.*

Det första sättet, med metoden `startDrag()`, är enklare men mer begränsat. När musknappen trycks ner anropas metoden `startDrag()` för visningsobjektet som ska dras. När musknappen släpps anropas metoden `stopDrag()`. Sprite-klassen definierar dessa två funktioner, och objektet som ska flyttas måste därför bestå av ett Sprite-objekt eller någon av dess underklasser.

```
// This code creates a mouse drag interaction using the startDrag()
// technique.
// square is a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Den här tekniken har en viktig begränsning: endast ett objekt i taget kan dras med `startDrag()`. Om ett visningsobjekt dras och metoden `startDrag()` anropas för ett annat visningsobjekt upphör det första visningsobjektet genast att följa musrörelsen. Om till exempel funktionen `startDragging()` ändras kommer endast objektet `circle` att dras, trots metodanropet `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Som en konsekvens av faktumet att endast ett objekt i taget kan dras via `startDrag()` kan metoden `stopDrag()` anropas för valfritt visningsobjekt och pågående objekt som dras kommer att stoppas.

Om du behöver dra fler än ett visningsobjekt, eller för att undvika en möjlig konflikt där fler än ett objekt kanske används med `startDrag()`, så är det bättre att använda den musföljande tekniken för att skapa drageffekten. När musknappen trycks ner och den här tekniken används agerar en funktion som avlyssnare för händelsen `mouseMove` på scenen. Med den här funktionen, som sedan anropas varje gång musen flyttas, hoppar objektet som dras till musens x,y-koordinat. När musknappen släpps slutar funktionen att agera som avlyssnare, vilket innebär att den inte längre anropas vid musrörelser och objektet upphör att följa efter musmarkören. Här följer ett kodexempel som visar tekniken:

```
// This code moves display objects using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Förutom att få ett visningsobjekt att följa efter musmarkören vill du kanske kunna flytta objektet längst fram på skärmen, så att det verkar flyta över alla andra objekt. Anta till exempel att du har två objekt, en cirkel och en kvadrat, som båda kan flyttas med musen. Om cirkeln råkar vara under fyrkanten på visningslistan och du klickar och drar cirkeln så att markören hamnar över fyrkanten så kommer cirkeln att läggas bakom fyrkanten, vilket bryter dra och släpp-illusionen. I stället kan du göra så att när cirkeln klickas, så flyttas den högst upp i visningslistan och på så vis läggs den framför annat innehåll.

Med följande kod (anpassad från det tidigare exemplet) kan du få två visningobjekt, en cirkel och en kvadrat, att flyttas med hjälp av musen. När musknappen trycks ner över någotdera objekt så flyttas det högst upp i scenens visningslista. Då hamnar det dragna objektet alltid överst. (Kod som är ny eller ändrad sedan föregående listning visas med fet stil.)

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    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);
```

Effekten kan utökas ytterligare, i till exempel spel där token eller kort flyttas mellan högar, genom att det dragna objektet läggs till scenens visningslista när det ”plockas upp”. Därefter lägger du objektet i en annan visningslista, t.ex. ”högen” där det lämnades, när musknappen släpps.

Slutligen, för att förstärka effekten, kan du använda ett skuggfilter på visningsobjektet när det klickas (och dras) och ta bort skuggan när objektet släpps. Mer information om hur du använder skuggfilter och andra visningsobjektfilter i ActionScript finns i ”Filtrera visningsobjekt” på sidan 268.

Panorering och rullning av visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du har ett visningsobjekt som är för stort för ytan där det ska presenteras kan du använda egenskapen `scrollRect` och definiera visningsytan för visningsobjektet. Dessutom, genom ändring av egenskapen `scrollRect` för att indata från användaren ska motsvaras kan innehållet panoreras från vänster till höger eller rullas uppåt och nedåt.

Egenskapen `scrollRect` är en instans i klassen Rectangle, där värdena som behövs för att definiera en rektangulär yta kombineras till ett enda objekt. För att först definiera visningsobjektets yta skapar du en ny Rectangle-instans och tilldelar den till visningsobjektets egenskap `scrollRect`. När du sedan vill rulla eller panorera läser du in egenskapen `scrollRect` till en separat Rectangle-variabel och ändrar önskad egenskap (till exempel Rectangle-instansens egenskap `x` för panorering eller `y` för bläddring.) Därefter omtilldelar du Rectangle-instansen till egenskapen `scrollRect` så att visningsobjektet aviseras om ändrat värde.

I nedanstående kod definieras den visningsbara ytan för ett TextField-objektet `bigText` som är för högt för att få plats inom SWF-filens gränser. När de båda knapparna `up` och `down` klickas anropas funktioner som gör att innehållet i TextField-objektet rullas uppåt eller nedåt genom att `y`-egenskapen i Rectangle-instansen `scrollRect` ändras.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

När du arbetar med egenskapen `scrollRect` för ett visningsobjekt är det bästa sättet att ange att Flash Player ska cachelagra visningsobjektets innehåll som en bitmapp med hjälp av egenskapen `cacheAsBitmap`. När du gör detta behöver inte Flash Player och AIR rita om hela innehållet i visningsobjektet varje gång det rullas. Den cachelagrade bitmappen kan i stället användas till att återge den nödvändiga delen direkt på bildskärmen. Mer information finns i ”Cachelagra visningsobjekt” på sidan 182.

Ändra storlek och skaländra objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns två sätt att mäta och ändra storlek på visningsobjekt, antingen via dimensionsegenskaperna (`width` och `height`) eller skalningsegenskaperna (`scaleX` och `scaleY`).

Alla visningsobjekt har egenskapen `width` och `height` som från början anges som storleken på objektet i antal pixlar. Du kan läsa in värdena för dessa egenskaper så att storleken på visningsobjektet mäts. Du kan även ange nya värden för att ändra objektets storlek enligt följande:

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Genom att ändra egenskaperna `height` eller `width` skaländras objektet, vilket innebär att innehållet sträcks ut eller trycks ihop för att passa in i den nya ytan. Om visningsobjektet innehåller enbart vektorformer kommer dessa att ritas om i den nya skalan, utan försämrad kvalitet. Alla grafikelement för bitmappar i visningsobjektet skaländras i stället för att ritas om. Ett digitalfoto, exempelvis, vars bredd och höjd ökas mer än de faktiska dimensionerna i bildens pixelinformation kommer att se taggigt ut.

När du ändrar egenskaperna `width` eller `height` i ett visningsobjekt uppdateras även egenskaperna `scaleX` och `scaleY`.

Obs! *TextField-objekt är ett undantag till det här skalningsbeteendet. Textfältens storlek måste ändras automatiskt och anpassas efter textbrytningar och teckenstorlekar. Därför återställs textfältens scaleX- eller scaleY-värden till 1 efter storleksändringen. Om du justerar scaleX- eller scaleY-värdena för ett TextField-objekt ändras och anpassas emellertid bredd- och höjdvärdena efter de skalningsvärden som du anger.*

Dessa egenskaper representerar den relativa storleken på visningsobjektet jämfört med den ursprungliga storleken. I egenskaperna `scaleX` och `scaleY` används decimalvärden som representerar procentandelar. Om till exempel visningsobjektets egenskap `width` har ändrats så att den utgör halva bredden mot ursprunglig storlek får objektets egenskap `scaleX` värdet `0,5`, vilket innebär 50 procent. Om höjden har dubblerats kommer egenskapen `scaleY` att få värdet `2`, vilket innebär 200 procent.

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

Storleksändringar är inte proportionella. M.a.o., om du ändrar egenskapen `height` för en fyrkant, men inte motsvarande `width`, så kommer proportionerna inte längre att vara desamma. Fyrkanten blir i stället en rektangel. Om du vill göra relativa storleksändringar på ett visningsobjekt kan du ange värdena för egenskaperna `scaleX` och `scaleY`, i stället för egenskaperna `width` eller `height`. Med följande kod ändras egenskapen `width` för visningsobjektet `square`. Därefter ändras den lodräta skalan (`scaleY`) för att motsvara den vågräta skalan, så att fyrkanten får en proportionell storlek.

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

Kontrollera förvrängning vid skalning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Normalt när ett visningsobjekt skaländras (till exempel vågrät utsträckning) fördelas den resulterande förvrängningen jämnt över hela objektet så att alla delar sträcks ut lika mycket. För grafik och designelement är detta precis vad du vill. Men ibland kan det vara önskvärt att kontrollera vilka delar av visningsobjektet som sträcks ut och vilka delar som ska förbli oförändrade. Ett vanligt exempel på detta är en knapp med en rektangulär form med rundade hörn. Med normal skalning sträcks hörnen ut, vilket innebär att hörnens radier ändras när knappstorleken ändras.

I det här fallet är det däremot önskvärt att kontrollera skalningen – att ange vissa ytor som ska skaländras (de raka sidorna och mitten) och ytor som inte ska ändras (hörnen). På så vis undviker man en förvrängning.

Du kan använda en skalning med 9 segment (Scale-9) för att skapa visningsobjekt där du har kontroll över hur objektet skaländras. Med en skalning med 9 segment delas visningsobjektet in i nio separata rektanglar (ett rutnät som är 3 x 3). Alla rektanglar behöver inte ha samma storlek – du anger var rutnätets linjer ska placeras. Allt innehåll som ligger inom rektanglarnas fyra hörn (till exempel de rundade hörnen på en knapp) kommer inte att sträckas ut eller tryckas ihop när visningsobjektet skaländras. De mittersta rektanglarna högst upp och längst ner skaländras vågrätt, men inte lodrätt. De mittersta rektanglarna till vänster och höger skaländras lodrätt, men inte vågrätt. Rektangeln i mitten skaländras både vågrätt och lodrätt.

Om du ska skapa ett visningsobjekt och du vill att ett visst innehåll aldrig skaländras, behöver du således bara se till att rutnätslinjerna i skalningen med 9 segment placeras så att innehållet avslutas med någon av hörnrektanglarna.

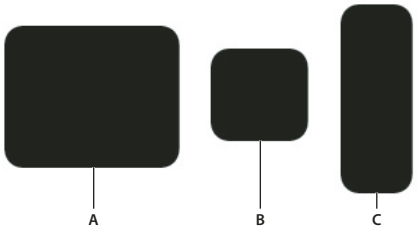
Om du i ActionScript anger ett värde för egenskapen `scale9Grid` för ett visningsobjekt aktiveras skalningen med 9 segment och storleken för rektanglarna anges i objektets rutnät. Du använder en instans av klassen Rectangle som värdet för egenskapen `scale9Grid` enligt följande:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

De fyra parametrarna i konstruktorn Rectangle är x- och y-koordinaterna, bredden och höjden. I det här exemplet placeras rektangelns översta vänstra hörn vid punkten x: 32, y: 27 på visningsobjektet `myButton`. Rektangeln är 71 pixlar bred och 64 pixlar hög (den högra kanten ligger på x-koordinaten 103 på visningsobjektet och nederkanten ligger på y-koordinaten 92 på visningsobjektet).

Den faktiska ytan som finns i regionen som definieras av Rectangle-instansen motsvarar mittenrektangeln i Scale-9-rutnätet. De övriga rektanglarna beräknas i Flash Player och AIR genom att sidorna i Rectangle-instansen förlängs:

När knappen i det här fallet skaländras uppåt eller nedåt kommer de rundade hörnen inte att påverkas, men övriga ytor justeras efter skalningen.



A. *myButton.width = 131;myButton.height = 106;* ***B.*** *myButton.width = 73;myButton.height = 69;* ***C.*** *myButton.width = 54;myButton.height = 141;*

Cachelagra visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Allt medan arbetet i Flash växer, oberoende av om du skapar ett program eller komplicerade skriptanimeringar, behöver du även tänka på prestanda och optimering. När du har ett innehåll som är statiskt (t.ex. en rektangulär Shape-instans) kommer innehållet inte att optimeras i Flash Player och AIR. När du ändrar rektangelns position ritas därför hela Shape-instansen om i Flash Player eller AIR.

Du kan cachelagra specifika visningsobjekt för att förbättra SWF-filens prestanda. Visningsobjektet är en *yta*, egentligen en bitmappsversion av instansens vektordata, som du troligtvis inte behöver ändra mycket under SWF-filens livslängd. Detta innebär att instanser där cachelagring är aktiverat inte ritas om kontinuerligt när SWF-filen spelas upp, vilket innebär att SWF-filen återges snabbare.

Obs! *Du kan uppdatera vektordata vilket medför att ytan skapas om på nytt. Detta innebär att vektordata som cachelagrats i ytan inte behöver vara desamma för hela SWF-filen.*

Inställning av visningsobjektets egenskap `cacheAsBitmap` till `true` får visningsobjektet att cachelagra en bitmappsrepresentation av sig själv. I Flash Player och AIR skapas ett ytobjekt för instansen som en cachelagrad bitmapp och inte som vektordata. Om du ändrar gränserna för visningsobjektet skapas ytan om i stället för att storleken ändras. Ytor kan kapslas i andra ytor. Bitmappen för den underordnade ytan kopieras till den överordnade ytan. Mer information finns i ”Aktivera bitmappscachning” på sidan 184.

Klassen DisplayObjects egenskaper `opaqueBackground` och `scrollRect` är båda relaterade till bitmappscachning med egenskapen `cacheAsBitmap`. Trots att dessa tre egenskaper är oberoende av varandra fungerar egenskaperna `opaqueBackground` och `scrollRect` bäst när ett objekt cachelagras som en bitmapp. Du ser prestandaförbättringar för egenskaperna `opaqueBackground` och `scrollRect` endast om du anger `cacheAsBitmap` till `true`. Mer information om rullning av visningsobjektinnehåll finns i ”Panorering och rullning av visningsobjekt” på sidan 178. Mer information om inställning av ogenomskinlig bakgrund finns i ”Inställning av en ogenomskinlig bakgrundsfärg” på sidan 185.

Mer information om alfakanalsmaskering, som kräver att du anger egenskapen `cacheAsBitmap` till `true`, finns i ”Maskering av visningsobjekt” på sidan 190.

Aktivera cachelagring

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du aktiverar cachelagring för ett visningsobjekt skapas en yta. Detta medför flera fördelar som att exempelvis komplexa vektoranimeringar återges snabbare. Det finns flera olika scenarier då du vill att cachelagring ska vara aktiverad. Det kan verka som om du alltid bör ha cachelagring aktiverad för att förbättra SWF-filens prestanda, men det finns situationer då cachelagring inte förbättrar prestandan eller till och med försämrar den. I detta avsnitt beskrivs tillfällen då cachelagring ska användas och när vanliga visningsobjekt ska användas.

Den övergripande prestandan för cachelagrade data beror på hur komplexa dina vektordata är i instansen, hur mycket data du ändrar och om du anger egenskapen `opaqueBackground` eller inte. Om du ändrar små områden är skillnaden mellan att använda en yta och att använda vektordata försumbar. Du kan testa båda scenarierna innan du distribuerar programmet.

Använda bitmapps-cachning

Följande exempel är typiska scenarier då du kan se stora fördelar med att aktivera bitmappscachning.

- Komplex bakgrundsbild: Ett program som innehåller en detaljerad och komplex bakgrundsbild av vektordata (kanske en bild där du använt kommandot för en bitmappskalkering eller teckning som du skapat i Adobe Illustrator®). Du kanske animerar tecken över bakgrunden, vilket gör att animeringen blir långsammare eftersom bakgrunden fortlöpande måste generera om vektorinformationen. Om du vill få förbättrad prestanda kan du ange egenskapen `opaqueBackground` för visningsobjektet i bakgrunden till `true`. Bakgrunden återges nu som en bitmapp och kan ritas om snabbare så att animeringen spelas upp mycket snabbare.
- Rullande textfält: Ett program som visar mycket text i ett rullande textfält. Du kan placera textfältet i ett visningsobjekt som du anger som rullningsbart med rullningsgränser (egenskapen `scrollRect`). Nu aktiveras snabb pixelrullning för den angivna instansen. När en användare rullar visningsobjektinstansen, förskjuts de rullande pixlarna uppåt i Flash Player eller AIR och det nyexponerade området genereras i stället för att hela textfältet genereras om.
- Fönstersystem: Ett program med komplexa system med överlappande fönster. Varje fönster kan öppnas och stängas (till exempel webbläsarfönster). Om du markerar varje fönster som en yta (anger att egenskapen `cacheAsBitmap` ska vara `true`) kommer varje fönster att isoleras och cachelagras. Användarna kan dra fönstren så att de överlappar varandra utan att vektorinnehållet för varje fönster behöver genereras om.
- Alfakanalsmaskering: När du använder alfakanalsmaskering måste du ange egenskapen `cacheAsBitmap` till `true`. Mer information finns i ”Maskering av visningsobjekt” på sidan 190.

Att aktivera bitmappscachning i alla dessa scenarier förbättrar svarstiderna och interaktiviteten i programmet tack vare att vektorgrafiken optimerats.

När du dessutom använder ett filter för ett visningsobjekt anges `cacheAsBitmap` automatiskt till `true`, även om du explicit anger det som `false`. Om du rensar alla filter för ett visningsobjekt ändras egenskapen `cacheAsBitmap` till vad den senast ställts in på.

Undvika bitmappscachning

Om du använder den här funktionen under fel omständigheter kan det påverka SWF-filens prestanda negativt. Tänk på följande riktlinjer när du använder bitmappscachning:

- Använd inte för många ytor (visningsobjekt med cachelagring aktiverad). För varje yta används mer minneskapacitet än för vanliga visningsobjekt, vilket innebär att du bara ska aktivera ytor när du vill förbättra återgivningsprestandan.

  En cachelagrad bitmapp kan använda betydligt mer minne än ett vanligt visningsobjekt. Om till exempel en Sprite-instans på scenen har storleken 250 x 250 pixlar kunde den vid en cachelagring ha använt 250 kB i stället för 1 kB vid en vanlig (ej cachelagrad) Sprite-instans.
- Undvik inzoomning i cachelagrade ytor. Om du använder för mycket bitmappscachning förbrukas en stor mängd minne (se punkten ovan), speciellt om du zoomar in i innehållet.
- Använd ytor för visningsobjektsinstanser som i huvudsak är statiska (ej animerade). Du kan dra eller flytta instansen, men instansinnehållet bör inte animeras eller ändras allt för mycket. (Animering eller innehållsändring är enklare att genomföra med en MovieClip-instans som innehåller animering eller en Video-instans.) Om du till exempel roterar eller omformar en instans, växlar instansen mellan ytan och vektorinformationen. Det är en svår bearbetning som påverkar SWF-filen negativt.
- Om du blandar ytor med vektordata ökas antalet bearbetningar som behöver göras i Flash Player och AIR (och ibland i datorn). Gruppera ytorna så mycket som möjligt när du skapar fönsterhanteringsprogram.
- Cachelagra inte objekt vars grafik ändras ofta. Varje gång du skalar, skevar eller roterar visningsobjektet, ändrar alfa- eller färgomformning, flyttar underordnade visningsobjekt eller ritar med egenskapen graphics, ritas bitmappscachen om. Om detta sker vid varje bildruta måste objektet ritas till en bitmapp och kopieras till scenen, vilket är mer krävande än att bara rita det ocachade objektet på scenen. Huruvida det är mest effektivt att använda cachning eller uppdateringsfrekvens beror på hur komplext och hur stort visningsobjektet är. Det kan du bara avgöra genom att testa det aktuella innehållet.

Aktivera bitmappscachning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill aktivera bitmappscachning för visningsobjekt anger du egenskapen `cacheAsBitmap` till `true`:

```
mySprite.cacheAsBitmap = true;
```

Sedan du angett egenskapen `cacheAsBitmap` till `true` kanske du noterar att visningsobjektsinstansen automatiskt pixelfästs till hela koordinater. När du testar SWF-filen upptäcker du att alla animeringar som utförs med komplexa vektorbilder återges mycket snabbare.

En yta (cachelagrad bitmapp) skapas inte även om `cacheAsBitmap` har satts till `true` när ett eller flera av följande alternativ inträffar:

- Bitmappen är större än 2 880 pixlar i höjd eller bredd.
- Bitmappen kan ej tilldelas (genererar ett fel om att minnet är slut).

Omformningsmatriser för cachelagrade bitmappar

Adobe AIR 2.0 och senare (mobilprofil)

I AIR-program för mobilenheter bör du ange egenskapen `cacheAsBitmapMatrix` när du anger egenskapen `cacheAsBitmap`. Genom att ange den här egenskapen kan du använda ett större antal omformningar på visningsobjektet utan att utlösa någon omgenerering.

```
mySprite.cacheAsBitmap = true;
mySprite.cacheAsBitmapMatrix = new Matrix();
```

När du anger den här matrix-egenskapen kan du använda följande extra omformningar på visningsobjektet utan att cacha om objektet:

- Flytta eller översätta utan pixelfästning
- Rotera
- Skala
- Skeva
- Ändra alfa (mellan 0 och 100 % genomskinlighet)

De här omformningarna används direkt på den cachelagrade bitmappen.

Inställning av en ogenomskinlig bakgrundsfärg

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan ställa in en ogenomskinlig bakgrund för ett visningsobjekt. När till exempel SWF-filen har en bakgrund som innehåller komplexa vektorbilder kan du välja en speciell färg för egenskapen `opaqueBackground` (vanligtvis samma färg som scenen). Färgen anges med en siffra (vanligtvis ett hexadecimalt färgvärde). Bakgrunden hanteras därefter som en bitmapp vilket hjälper till att optimera prestandan.

När du anger `cacheAsBitmap` till `true`, och också anger en specifik bakgrundsfärg med egenskapen `opaqueBackground`, tillåter egenskapen `opaqueBackground` att den interna bitmappen blir ogenomskinlig och återges snabbare. Om du inte anger `cacheAsBitmap` till `true`, lägger egenskapen `opaqueBackground` till en ogenomskinlig rektangulär vektorfigur till bakgrunden av visningsobjektet. Ingen bitmapp skapas automatiskt.

I exemplet nedan visas hur du ställer in bakgrunden för ett visningsobjekt för att optimera prestandan:

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

I det här fallet anges bakgrundsfärgen för formen `myShape` som röd (`0xFF0000`). Anta att Shape-instansen innehåller en bild av en grön triangel, på en scen med vit bakgrund. Då skulle den visas som en grön triangel med rött i den tomma ytan i Shape-instansens begränsningsram (rektangeln som helt innesluter formen).

Den här koden skulle givetvis verka mer logisk om den användes med en scen med en heltäckande röd bakgrund. Med en annan färgad bakgrund skulle den färgen anges i stället. I till exempel en SWF-fil med en vit bakgrund skulle antagligen egenskapen `opaqueBackground` anges med `0xFFFFFF`, eller helt vit.

Använda blandningslägen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Blandningslägen inkluderar kombination av färgerna i en bild (basbilden) med färgerna i en annan bild (blandningsbilden) för att skapa en tredje bild; den resulterande bilden är den som faktiskt visas på bildskärmen. Varje pixelvärde i en bild bearbetas med motsvarande pixelvärde från den andra bilden för att skapa ett pixelvärde för samma position i resultatet.

Alla visningsobjekt har egenskapen `blendMode` som kan anges med ett av följande blandningslägen. Dessa är konstanter som finns definierade i klassen BlendMode. Alternativt kan du använda strängvärden (inom parentes) som motsvarar konstanternas faktiska värden.

- `BlendMode.ADD` (`"lägg till"`): Används vanligen för att skapa en animerad blixtupplösningseffekt mellan två bilder.
- `BlendMode.ALPHA` (`"alpha"`): Används oftast för att använda förgrundens genomskinlighet på bakgrunden. (Stöds inte vid GPU-återgivning.)
- `BlendMode.DARKEN` (`"darken"`): Används oftast för att lägga text ovanpå något annat. (Stöds inte vid GPU-återgivning.)
- `BlendMode.DIFFERENCE` (`"difference"`): Används vanligen för att skapa livfullare färger.
- `BlendMode.ERASE` (`"erase"`): Används vanligen för att klippa ut (sudda ut) delar av bakgrunden med hjälp av förgrundsalfa. (Stöds inte vid GPU-återgivning.)
- `BlendMode.HARDLIGHT` (`"hardlight"`): Används oftast för att skapa skuggningseffekter. (Stöds inte vid GPU-återgivning.)
- `BlendMode.INVERT` (`"invert"`): Används för att invertera bakgrunden.
- `BlendMode.LAYER` (`"layer"`): Används för att tvinga en tillfällig buffert att skapas för förkomponering av ett visst visningsobjekt. (Stöds inte vid GPU-återgivning.)
- `BlendMode.LIGHTEN` (`"lighten"`): Används oftast för att lägga text ovanpå något annat. (Stöds inte vid GPU-återgivning.)
- `BlendMode.MULTIPLY` (`"multiplicera"`): Används vanligen för att skapa skuggor och djupeffekter.
- `BlendMode.NORMAL` (`"normal"`): Används för att specificera att pixelvärdena för blandningsbilden åsidosätter värdena för basbilden.
- `BlendMode.OVERLAY` (`"overlay"`): Används oftast för att skapa skuggningseffekter. (Stöds inte vid GPU-återgivning.)
- `BlendMode.SCREEN` (`"raster"`): Används vanligen för att skapa markeringar och linsöverstrålning.
- `BlendMode.SHADER` (`"shader"`): Används för att ange att en Pixel Bender-skuggning nyttjas för att skapa en anpassad blandningseffekt. Mer information om hur du använder skuggningar finns i ”Arbeta med Pixel Bender-skuggningar” på sidan 301. (Stöds inte vid GPU-återgivning.)
- `BlendMode.SUBTRACT` (`"subtrahera"`): Används vanligen för att skapa en animerad mörkerupplösningseffekt mellan två bilder.

Justera färger i DisplayObject

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda metoderna för klassen ColorTransform (flash.geom.ColorTransform) för att justera färgen på ett visningsobjekt. Alla visningsobjekt har egenskapen `transform` som är en instans av klassen Transform och innehåller information om olika omformningar som används för visningsobjektet (till exempel rotation, ändringar i skala eller position). Utöver den här informationen om geometriska omformningar innehåller klassen Transform även egenskapen `colorTransform`, vilket är en instans av klassen ColorTransform och ger åtkomst till färgjusteringar för visningsobjektet. För åtkomst av informationen för färgomformning för ett visningsobjekt använder du till exempel följande kod:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

När du har skapat instansen ColorTransform kan du läsa egenskapsvärdena och ta reda på vilka färgomformningar som redan används, eller också kan du ange dessa värden för att göra visningsobjektets färgändringar. För att uppdatera visningsobjektet efter gjorda ändringar måste du omtilldela ColorTransform-instansen tillbaka till egenskapen `transform.colorTransform`.

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

Inställningar av färgvärden med kod

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskapen `color` för klassen ColorTransform kan användas vid tilldelning av ett specifikt färgvärde för rött, grönt eller blått (RGB) för visningsobjektet. I följande exempel används egenskapen `color` för att ändra färgen på visningsobjektet `square` till blått, när användaren klickar på knappen `blueBtn`:

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Observera, att när du ändrar visningsobjektets färg med egenskapen `color` ändras objektets färg helt, oavsett om det tidigare hade flera färger. Om till exempel ett visningsobjekt innehåller en grön cirkel med svart text över och du anger egenskapen `color` för objektets ColorTransform-instans med en röd nyans, kommer hela objektet (cirkel och text) att bli rött och texten går inte att urskilja från resten av objektet.

Ändra färg och intensitetseffekter med kod

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Anta att du har ett visningsobjekt med flera färger (till exempel ett digitalt foto) och du inte helt vill ändra färg på objektet. Du vill bara justera färgen på visningsobjektet baserat på befintliga färger. I det här fallet finns det i klassen ColorTransform flera multiplikator- och offsetegenskaper som kan användas. Multiplikatoregenskaperna som heter `redMultiplier`, `greenMultiplier`, `blueMultiplier` och `alphaMultiplier` fungerar som färgade fotofilter (eller solglasögon) och förstärker eller försvagar vissa färger i visningsobjektet. Offsetegenskaperna (`redOffset`, `greenOffset`, `blueOffset` och `alphaOffset`) kan användas för att lägga till extra mycket av vissa färger till ett objekt eller ange ett minimumvärde som en specifik färg kan ha.

Dessa multiplikator- och offsetegenskaper är identiska med de avancerade färginställningarna som finns för filmklippssymboler i utvecklingsverktyget när du väljer Avancerat i snabbmenyn Färg i egenskapsinspektören.

Med följande kod läser du in en JPEG-bild och använder en färgomformning, vilket ändrar de röda och gröna kanalerna när muspekaren flyttar längs x- och y-axeln. I det här fallet, eftersom inga offsetvärden har angivits, kommer färgvärdet för respektive färgkanal som visas på bildskärmen att vara en procentandel av det ursprungliga färgvärdet i bilden. Detta innebär att det mest röda eller gröna som visas i en given pixel kommer att få den ursprungliga mängden rött eller grönt i pixeln.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

Rotera objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Visningsobjekt kan roteras med hjälp av egenskapen `rotation`. Du kan läsa detta värde för att ta reda på om ett objekt har roterats. Om du vill rotera objektet kan du ange egenskapen med en siffra (antal grader) som motsvarar hur mycket objektet ska roteras. Med den här kodraden roteras objektet `square` 45 grader (en åttondel av ett helt varv):

```
square.rotation = 45;
```

Alternativt kan du rotera ett visningsobjekt med hjälp av en omformningsmatris som beskrivs i ”Arbeta med geometri” på sidan 210.

Tona objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan kontrollera genomskinligheten för ett visningsobjekt så att det blir delvis (eller helt) genomskinligt eller också kan du ändra genomskinligheten för att objektet ska tonas in eller ut. Klassen DisplayObjects egenskap `alpha` anger visningsobjektets genomskinlighet (eller, mer korrekt, opaciteten). Egenskapen `alpha` kan anges med ett värde mellan 0 och 1, där 0 är helt genomskinligt och 1 är helt ogenomskinligt. Med följande kodrader blir objektet `myBall` delvis (50 procent) genomskinligt vid en musklickning.

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

Du kan även ändra genomskinligheten för ett visningsobjekt via de färgjusteringar som finns i klassen ColorTransform. Mer information finns i ”Justera färger i DisplayObject” på sidan 187.

Maskering av visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda ett visningsobjekt som en mask för att skapa ett hål genom vilket innehållet från ett annat visningsobjekt blir synligt.

Definiera en mask

För att ange att ett visningsobjekt ska vara mask för ett annat visningsobjekt anger du maskobjektet som `mask`-egenskapen för visningsobjektet som ska maskeras:

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

Det maskerade visningsobjektet visas under alla ogenomskinliga områden av visningsobjektet som fungerar som masken. Med följande kod skapas en Shape-instans med en röd fyrkant som är 100 x 100 pixlar och en Sprite-instans med en blå cirkel som har en radie på 25 pixlar. När cirkeln klickas anges den som masken för fyrkanten, så att den enda delen av fyrkanten som visas är den del som täcks av den heltäckande delen av cirkeln. M.a.o. visas endast en röd cirkel.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
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);

// Draw a circle and add it to the display list.
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);
```

Visningsobjektet som fungerar som en mask kan vara dragbart, animerat, dynamiskt storleksändrat och kan använda separata former inom en enstaka mask. Maskens visningsobjekt behöver inte läggas till i visningslistan. Om du däremot vill att maskobjektet skaländras när Stage skaländras eller om du vill aktivera användarinteraktion med masken (till exempel användarkontrollerad dragning och storleksändring) så måste maskobjektet läggas till i visningslistan. Det faktiska z-indexet (överst till underst-ordning) för visningsobjektet spelar ingen roll, bara maskobjektet finns med i visningslistan. (Maskobjektet visas inte på bildskärmen annat än som en mask.) Om maskobjektet är en MovieClip-instans med flera bildrutor spelas alla bildrutor upp i tidslinjen, på samma sätt som om det inte skulle vara en mask. Du kan ta bort en mask genom att ange egenskapen `mask` med `null`:

```
// remove the mask from mySprite
mySprite.mask = null;
```

Du kan inte använda en mask för att maskera en annan mask. Du kan inte välja egenskapen `alpha` för ett maskerat visningsobjekt. Endast fyllningar används i visningsobjekt som används som en mask, linjer ignoreras.

AIR 2

Om ett maskerat visningsobjekt cachelagras genom inställning av egenskaperna `cacheAsBitmap` och `cacheAsBitmapMatrix` måste masken vara underordnad det maskerade visningsobjektet. Om det maskerade visningsobjektet är underordnat en visningsobjektsbehållare som är cachelagrad måste på samma sätt både masken och visningsobjektet vara underordnade den behållaren. Om det maskerade objektet är underordnat fler än en cachelagrad visningsobjektsbehållare måste masken vara underordnad den cachelagrade behållare som är närmast det maskerade objektet i visningslistan.

Maskning av enhetsteckensnitt

Du kan använda ett visningsobjekt för att maskera text angivet i ett enhetsteckensnitt. När du använder ett visningsobjekt för att maskera en textuppsättning i ett enhetsteckensnitt, används den rektangulärt avgränsade rutan som maskeringsfigur. D.v.s., att om du skapar en icke-rektangulär visningsobjektmask för enhetsteckensnittstext, kommer masken som visas i SWF-filen att ha formen av den rektangulärt avgränsade rutan av masken, inte formen av själva masken.

Alfakanalsmaskering

Alfakanalsmaskering stöds om bitmappscachning används både i masken och i de maskerade visningsobjekten enligt följande:

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

En tillämpning av alfakanalsmaskering är användning av ett filter på maskobjektet oberoende av ett filter som tillämpas på det maskerade visningsobjektet.

I följande exempel läses en extern bildfil in till scenen. Bilden (egentligen Loader-instansen den läses in till) blir det visningsobjekt som är maskerat. En övertoningsoval (heltäckande mittpunkt som tonas ut till genomskinlighet vid kanterna) ritas över bilden; detta utgör alfamasken. Båda visningsobjekten har aktiverad bitmappscachning. Ovalen anges som en mask för bilden och görs sedan dragbar.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
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();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

Animera objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Animering är att tillsätta en rörelse eller att skapa en ändring med tiden. Animering via skript är en grundläggande del i videospel och används ofta för extraeffekter och interaktiva hänvisningar till andra program.

Den grundläggande idén bakom animering via skript är att en ändring behöver inträffa och att ändringen måste delas upp i inkrement under en viss tid. Det är enkelt att få en upprepning i ActionScript med hjälp av en vanlig programslinga. Men med en slinga körs alla iterationer innan visningen uppdateras. När du ska skapa skriptanimering måste du skriva ett ActionScript där något utförs upprepade gånger under en tid och skärmbilden uppdateras vid varje körning.

Tänk dig till exempel att du vill skapa en enkel animering, som att få en boll att rulla över skärmbilden. ActionScript innehåller en enkel mekanism som gör att du kan följa tidsförloppet och uppdatera skärmbilden enligt detta, vilket innebär att du kan skriva kod som flyttar bollen en liten bit i taget ända tills destinationen uppnås. Efter varje rörelse uppdateras visningen så att användaren ser föremålets rörelse på scenen.

Ur en praktisk synvinkel är det logiskt att synkronisera skriptanimeringar med SWF-filens bildrutehastighet (d.v.s. att göra en animeringsändring varje gång en ny bildruta visas) eftersom detta definierar hur ofta Flash Player eller AIR uppdaterar skärmen. Alla visningsobjekt innehåller händelsen `enterFrame` som skickas enligt SWF-filens bildrutehastighet – en händelse per bildruta. De flesta utvecklare som skapar skriptanimering använder händelsen `enterFrame` för att skapa rörelser som upprepas under tiden. Du kan skriva kod som avlyssnar händelsen `enterFrame` så att den animerade bollen flyttas en viss sträcka för varje bildruta, och när skärmbilden uppdateras (varje bildruta) ritas bollen om i sin nya placering och skapar en rörelse.

Obs! *Ett annat sätt att utföra en åtgärd upprepade gånger under en tid är med hjälp av klassen Timer. En Timer-instans utlöser en händelseavisering varje gång som en viss tidsperiod har passerat. Du kan skriva kod som utför animering genom hantering av klassen Timers timerhändelse, där du anger ett kort tidsintervall (en del av en sekund). Mer information om hur du använder klassen Timer finns i ”Kontrollera tidsintervall” på sidan 4.*

I följande exempel skapas en cirkelformad Sprite-instans, `circle`, på scenen. När användaren klickar på cirkeln startas en skriptanimeringssekvens som får `circle` att tona ut (egenskapen `alpha` stegas ner) tills den är helt genomskinlig:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
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);
```

När användaren klickar på cirkeln används funktionen `fadeCircle()` som avlyssnare för händelsen `enterFrame`, vilket innebär att den anropas vid varje bildruta. Funktionen tonar ut `circle` genom att egenskapen `alpha` ändras, så att cirkelns `alpha` minskas med 0,05 (5 procent) för varje bildruta och skärmbilden uppdateras. Efter en stund när värdet `alpha` har blivit 0 (`circle` är helt genomskinlig) tas funktionen `fadeCircle()` bort som händelseavlyssnare och animeringen avslutas.

Samma kod kan användas för att exempelvis skapa animerad rörelse i stället för uttoning. Genom att ersätta en annan egenskap med `alpha` i funktionen som är händelseavlyssnaren `enterFrame` kommer i stället denna egenskap att animeras. Genom att till exempel ändra raden

```
    circle.alpha -= .05;
```

till denna kod

```
    circle.x += 5;
```

animeras egenskapen `x`, vilket gör att cirkeln flyttas till höger över scenen. Villkoret som avslutar animeringen kan ändras så att animeringen avslutas (d.v.s. avlyssnaren `enterFrame` avbryts) när önskad x-koordinat uppnås.

Scenorientering

AIR 2.0 och senare

Mobilenheter orienterar vanligtvis om användargränssnittet för att skärmen ska visas på rätt håll när användaren roterar enheten. Om du aktiverar automatisk orientering i programmet ser enheten till att skärmen visas på rätt håll, men du måste själv se till att ditt innehåll ser bra ut när scenens orientering (proportioner) ändras. Om du inaktiverar automatisk orientering ändrar enheten inte skärmens orientering, om du inte ändrar den manuellt.

AIR-program används på ett flertal olika mobilenheter och operativsystem. Det underliggande orienteringsbeteendet kan variera mellan olika operativsystem, till och med mellan olika enheter med samma operativsystem. En enkel designstrategi, som fungerar bra för alla enheter och operativsystem, är att aktivera automatisk orientering och lyssna efter `resize`-händelser för scenen för att avgöra när programmets layout behöver uppdateras.

Ett alternativ, om ditt program bara har stöd för antingen liggande eller stående orientering, är att inaktivera automatisk orientering och ange de proportioner som stöds i AIR-programmets beskrivningsfil. På så sätt får programmet ett konsekvent beteende och den bästa orientering för de valda proportionerna används. Om du till exempel anger liggande orientering väljs en orientering som passar enheter med liggande läge och utfällbara tangentbord.

Hämta scenens aktuella orientering och proportioner

Orienteringen rapporteras i förhållande till enhetens normala position. På de flesta enheter finns det en tydlig, upprätt position. Den här positionen är *standardorienteringen*. Övriga tre orienteringar är då *roterad åt vänster*, *roterad åt höger* och *upp-och-ned*. Klassen StageOrientation definierar strängkonstanter som används när orienteringsvärden anges eller jämförs.

Klassen Stage definierar två egenskaper som rapporterar om orienteringen:

- Stage.deviceOrientation – Rapporterar om enhetens fysiska orientering i förhållande till standardpositionen.

  Obs! *deviceOrientation är inte alltid tillgängligt när programmet först startas eller när enheten ligger ned. I så fall rapporteras enhetens orientering som okänd.*

- Stage.orientation – Rapporterar om scenens orientering i förhållande till standardpositionen. När automatisk orientering är aktiverat roteras scenen i motsatt riktning som enheten för att förbli upprätt. Den högra och vänstra positionen som rapporteras av egenskapen `orientation` är alltså motsatserna till de som rapporteras av egenskapen `deviceOrientation`. Om `deviceRotation` till exempel rapporterar *roterad åt höger* rapporterar `orientation` i stället *roterad åt vänster*.

Scenens proportioner kan härledas genom att helt enkelt jämföra scenens aktuella bredd och höjd:

```
var aspect:String = this.stage.stageWidth >= this.stage.stageHeight ?
StageAspectRatio.LANDSCAPE : StageAspectRatio.PORTRAIT;
```

Automatisk orientering

När automatisk orientering är aktiverat, och användaren roterar enheten, orienteras hela användargränssnittet om, inklusive aktivitetsfältet och ditt program, av operativsystemet. Följaktligen ändras scenens proportioner från stående till liggande eller tvärtom. När proportionerna ändras, ändras även scenens dimensioner.

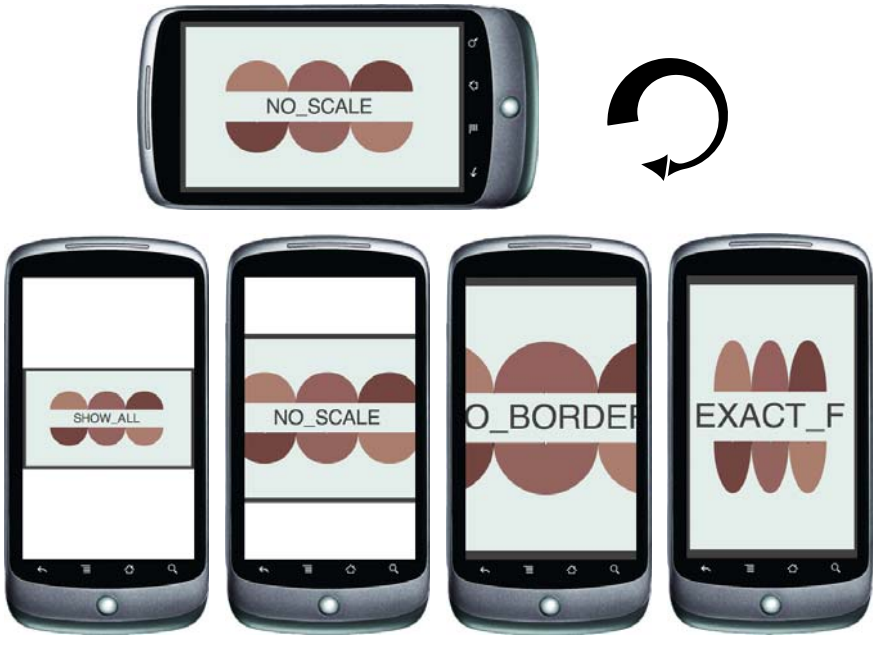
Du kan aktivera eller inaktivera automatisk orientering vid körning genom att ange egenskapen `autoOrients` för scenen som `true` eller `false`. Du kan ange det inledande värdet på den här egenskapen i AIR-programbeskrivningen med elementet `<autoOrients>`. (Tänk på att i tidigare versioner än AIR 2.6 är `autoOrients` en skrivskyddad egenskap, som bara kan anges i programbeskrivningsfilen.)

Om du anger proportionerna för liggande och stående samt dessutom aktiverar automatisk orientering, kommer AIR att begränsa den automatiska orienteringen till de angivna proportionerna.

Ändringar av scendimensioner

När scenens dimensioner ändras skalas sceninnehållet och placeras om i enlighet med egenskaperna `scaleMode` och `align` för Stage-objektet. För det mesta blir resultatet inte så bra om du förlitar dig på det automatiska beteende som `scaleMode`-inställningarna för scenen ger. Du måste i stället göra om layouten eller rita om grafiken och komponenterna för att skapa stöd för mer än en proportion. (Med flexibel layoutlogik fungerar programmet även bättre på enheter med olika skärmstorlekar och proportioner.)

Följande bild visar effekten av de olika `scaleMode`-inställningarna när en vanlig mobilenhet roteras:



Rotation från liggande till stående orientering

Bilden visar det skalningsbeteende som används vid rotation från liggande till stående orientering med olika skalningslägen. Rotation från stående till liggande orientering ger upphov till liknande effekter.

Orienteringsändringshändelser

Stage-objektet skickar två sorters händelser som du kan använda för att identifiera och reagera på orienteringsändringar. Både `resize`- och `orientationChange`-händelser för scenen skickas när automatisk orientering aktiveras.

Det är bäst att använda *resize*-händelsen när du förlitar dig på automatisk orientering för att hålla skärmen upprätt. När scenen skickar en `resize`-händelse ritar du om innehållet eller ändrar dess layout efter behov. `resize`-händelsen skickas bara när scenens skalningsläge är inställt på `noScale`.

Du kan också använda `orientationChange`-händelsen för att identifiera orienteringsändringar. `orientationChange`-händelsen skickas bara när automatisk orientering är aktiverat.

Obs! *På vissa mobila plattformar skickar scenen en `orientationChanging`-händelse, som kan avbrytas, innan resize- eller orientationChange-händelser skickas. Eftersom händelsen inte stöds på alla plattformar bör du inte förlita dig på den.*

Manuell orientering

AIR 2.6 och senare

Du kan styra scenens orientering med metoderna `setOrientation()` och `setAspectRatio()` för Stage-objektet.

Ange scenens orientering

Du kan ange scenens orientering vid körning med metoden `setOrientation()` för Stage-objektet. Använd de strängkonstanter som definieras i klassen StageOrientation för att ange önskad orientering:

```
this.stage.setOrientation( StageOrientation.ROTATED_RIGHT );
```

Alla enheter och operativsystem har inte stöd för alla orienteringar. Till exempel saknar Android 2.2 stöd för att via programmet välja en vänsterroterad orientering på enheter med stående standardformat och saknar helt stöd för upp och ned-orientering. Egenskapen `supportedOrientations` för scenen tillhandahåller en lista över de orienteringar som kan skickas till metoden `setOrientation()`:

```
var orientations:Vector.<String> = this.stage.supportedOrientations;
for each( var orientation:String in orientations )
{
    trace( orientation );
}
```

Ange scenens proportioner

Om scenens proportioner är viktigast kan du ange proportionerna som stående eller liggande. Du kan ange proportionerna antingen i AIR-programbeskrivningen eller, vid körning, med metoden `setAspectRatio()` för Stage-objektet:

```
this.stage.setAspectRatio( StageAspectRatio.LANDSCAPE );
```

En av de två möjliga orienteringarna för den angivna proportionen väljs automatiskt. Detta kanske inte matchar enhetens aktuella orientering. Standardorienteringen väljs till exempel framför upp och ned-orientering (AIR 3.2 och tidigare) och den orientering som passar för utfällbara tangentbord väljs framför den motsatta orienteringen.

(AIR 3.3 och senare) Från och med AIR 3.3 (SWF-version 16) kan du även använda konstanten `StageAspectRatio.ANY`. Om `Stage.autoOrients` är inställt på `true` och du anropar `setAspectRatio(StageAspectRatio.ANY)` kan programmet orienteras om till alla orienteringslägen (liggande vänster, liggande höger, stående och stående upp och ned). En annan nyhet i AIR 3.3 är att proportionen är bestående och att ytterligare rotation av enheten är begränsad till den specifika orienteringen.

Exempel: Ange scenens orientering så att den matchar enhetens

Följande exempel visar en funktion som uppdaterar scenens orientering så att den matchar enhetens aktuella orientering. Egenskapen `deviceOrientation` för scenen anger enhetens fysiska orientering, även när automatisk orientering är inaktiverat.

```
function refreshOrientation( theStage:Stage ):void
{
    switch ( theStage.deviceOrientation )
    {
        case StageOrientation.DEFAULT:
            theStage.setOrientation( StageOrientation.DEFAULT );
            break;
        case StageOrientation.ROTATED_RIGHT:
            theStage.setOrientation( StageOrientation.ROTATED_LEFT );
            break;
        case StageOrientation.ROTATED_LEFT:
            theStage.setOrientation( StageOrientation.ROTATED_RIGHT );
            break;
        case StageOrientation.UPSIDE_DOWN:
            theStage.setOrientation( StageOrientation.UPSIDE_DOWN );
            break;
        default:
            //No change
    }
}
```

Orienteringsändringen är asynkron. Du kan lyssna efter `orientationChange`-händelsen som skickas av scenen för att identifiera att ändringen har slutförts. Om en orientering inte stöds på en enhet misslyckas `setOrientation()`-anropet utan att något fel genereras.

Dynamisk inläsning av visningsinnehåll

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan läsa in någon av följande externa visningsresurser till ett program i ActionScript 3.0:

- En SWF-fil som har skapats i ActionScript 3.0 – den här filen kan ha klassen Sprite, MovieClip eller någon annan klass som utökar Sprite. I AIR-program på iOS är det bara SWF-filer som inte innehåller ActionScript-bytekod som kan läsas in. Det innebär att SWF-filer som innehåller inbäddade data, t.ex. bilder och ljud, kan läsas in, men inte SWF-filer som innehåller körbar kod.
- En bildfil – dessa är JPG-, PNG- och GIF-filer.
- En AVM1 SWF-fil – detta är en SWF-fil skriven i ActionScript 1.0 eller 2.0. (stöds inte i AIR-mobilprogram)

Inläsning av resurserna görs via klassen Loader.

Inläsning av visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Loader-objekt används för att läsa in SWF-filer och grafikfiler till ett program. Klassen Loader är en underklass till klassen DisplayObjectContainer. Ett Loader-objekt kan innehålla endast ett underordnat visningsobjekt i sin visningslista – visningsobjektet som motsvarar SWF-filen eller grafikfilen som läses in. När du lägger ett Loader-objekt till visningslistan, som i nedanstående kod, kan du även lägga till det inlästa underordnade visningsobjektet till visningslistan efter inläsningen:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

När SWF-filen eller bilden är inläst kan du flytta visningsobjektet till en annan behållare för visningsobjekt, till exempel DisplayObjectContainer-objektet `container` i detta exempel:

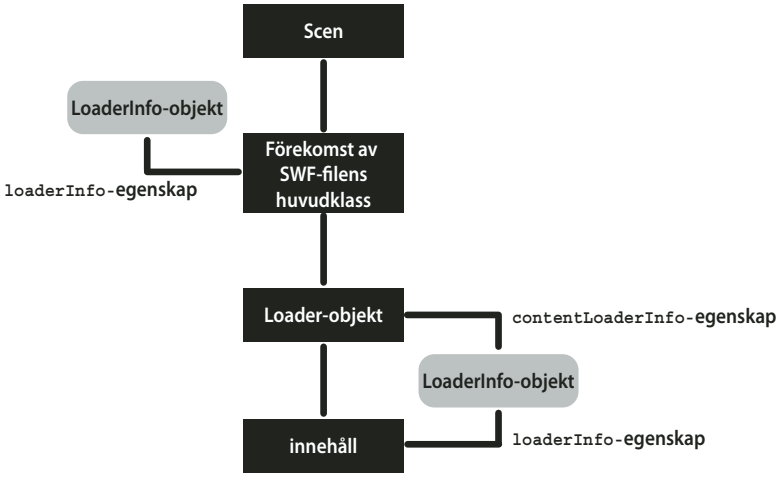
```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

Övervaka inläsningsförloppet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När filen har börjat läsas in skapas objektet LoaderInfo. Ett LoaderInfo-objekt innehåller information om inläsningsförloppet, inläsarens och utläsarens URL-adresser, det sammanlagda antalet byte för mediet samt mediets nominella höjd och bredd. Ett LoaderInfo-objekt skickar även händelser som övervakar förloppet av inläsningen.

I följande diagram visas hur LoaderInfo-objektet kan användas – för huvudklassinstansen i SWF-filen, för ett Loader-objekt och för ett objekt inläst via Loader-objektet:



Objektet LoaderInfo är tillgängligt som egenskap för både Loader-objektet och det inlästa visningsobjektet. Så snart inläsningen påbörjas blir LoaderInfo-objektet tillgängligt via egenskapen `contentLoaderInfo` i Loader-objektet. När visningsobjektet är inläst blir LoaderInfo-objektet tillgängligt även som egenskap för det inlästa visningsobjektet via egenskapen `loaderInfo`. Egenskapen `loaderInfo` för det inlästa visningsobjektet refererar till samma LoaderInfo-objekt som egenskapen `contentLoaderInfo` för Loader-objektet. Ett LoaderInfo-objekt delas m.a.o. mellan ett inläst objekt och det Loader-objekt som läste in det (mellan inläsare och utläsare).

För att få tillgång till egenskaper för inläst innehåll måste du lägga till en händelseavlyssnare till LoaderInfo-objektet enligt följande:

```
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;
}
```

Mer information finns i ”Hantera händelser” på sidan 125.

Ange innehåll för inläsning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du läser in en extern fil till Flash Player via metoderna `load()` eller `loadBytes()` i klassen Loader kan du välja att ange parametern `context`. Denna parameter är objektet LoaderContext.

Klassen LoaderContext innehåller tre egenskaper som du anger för att definiera hur inläst innehåll ska användas:

- `checkPolicyFile`: Använd den här egenskapen endast när du läser in en bildfil (inte en SWF-fil). Om du anger egenskapen till `true` kontrolleras om det finns en principfil på den ursprungliga servern (se ”Webbplatsinställningar (principfiler)” på sidan 1056). Detta är endast nödvändigt för innehåll som kommer från andra domäner än den som SWF-filen med Loader-objektet kommer från. Om servern ger Loader-domänen tillstånd så kan ActionScript från SWF-filer i domänen få dataåtkomst till den inlästa bilden. Du kan m.a.o. använda kommandot `BitmapData.draw()` för att få åtkomst till data i den inlästa bilden.

  Observera att en SWF-fil från andra domäner än den som Loader-objektet kan anropa `Security.allowDomain()` för att tillåta en specifik domän.

- `securityDomain`: Använd bara den här egenskapen när du läser in en SWF-fil (inte en bild). Ange den här för en SWF-fil från en annan domän än den som filen med Loader-objektet har. När du anger detta alternativ görs en sökning efter en korsdomänprincip och om sådan finns kan inläst SWF-innehåll korsskriptas av SWF-filer i de domäner som ges tillstånd med korsdomänprincipen. Du kan ange `flash.system.SecurityDomain.currentDomain` som denna parameter.

- `applicationDomain`: Använd bara den här egenskapen när du läser in en SWF-fil som är skriven i ActionScript 3.0 (inte en bild eller SWF-fil som är skriven i ActionScript 1.0 eller 2.0). Vid inläsning av filen kan du ange att filen ska ingå i samma programdomän som Loader-objektet. Detta gör du genom att ange parametern `applicationDomain` med `flash.system.ApplicationDomain.currentDomain`. Genom att placera den inlästa SWF-filen i samma programdomän får du direktåtkomst till klasserna. Detta är praktiskt om du läser in en SWF-fil som innehåller inbäddade medier, som du får tillgång till via associerade klassnamn. Mer information finns i "Arbeta med programdomäner" på sidan 148.

Här är ett exempel på sökning efter en principfil vid inläsning av en bitmapp från en annan domän:

```
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);
```

Här visas ett exempel på sökning efter en principfil vid inläsning av en SWF-fil från en annan domän, för placering av filen i samma säkerhetssandlåda som Loader-objektet. Med koden läggs dessutom klasserna i den inlästa SWF-filen till i samma programdomän som Loader-objektet:

```
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);
```

Mer information finns i klassbeskrivningen för LoaderContext i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Läsa in SWF-filer i AIR for iOS

Adobe AIR 3.6 och senare, endast iOS

På iOS-enheter finns det begränsningar för inläsning och kompilering av kod vid körning. På grund av dessa begränsningar finns det vissa skillnader när det gäller hur externa SWF-filer läses in till programmet på sådana enheter:

- Alla SWF-filer som innehåller ActionScript-kod måste inkluderas i programpaketet. Inga SWF-filer med kod kan läsas in från externa källor, till exempel via ett nätverk. Som en del av programpaketeringen kompileras all ActionScript-kod i alla SWF-filer i programpaketet till systemspecifik kod för iOS-enheter.
- Det går inte att läsa in, ta bort och sedan läsa in en SWF-fil igen. Om du försöker göra det genereras ett fel.
- Beteendet med att läsa in till minnet och sedan ta bort det därifrån är samma som på skrivbordsplattformar. Om du läser in en SWF-fil och sedan tar bort den tas alla visuella resurser som finns i SWF-filen bort från minnet. Däremot bevaras alla klassreferenser till ActionScript-klasser i den inlästa SWF-filen i minnet och dessa går att komma åt via ActionScript-koden.
- Alla inlästa SWF-filer måste läsas in till samma programdomän som SWF-huvudfilen. Detta är inte standardbeteende, och därför måste du för varje SWF-fil som läses in skapa ett LoaderContext-objekt som anger huvudprogramdomänen och skicka det LoaderContext-objektet till Loader.load()-metodanropet. Om du försöker läsa in en SWF-fil till en annan programdomän än SWF-filens huvudprogramdomän genereras ett fel. Detta gäller även om den inlästa SWF-filen bara innehåller visuella resurser och ingen ActionScript-kod.

  Följande exempel visar den kod som används för att läsa in en SWF-fil från programpaketet till SWF-huvudfilens programdomän:

```
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("swfs/SecondarySwf.swf");
var loaderContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain,
null);
loader.load(url, loaderContext);
```

En SWF-fil som bara innehåller resurser och ingen kod kan läsas in från programpaketet eller via ett nätverk. SWF-filen måste i båda fallen ändå läsas in till huvudprogramdomänen.

För tidigare AIR-versioner än AIR 3.6 rensas all kod ut från alla SWF-filer utom programmets SWF-huvudfil under kompileringen. SWF-filer som bara innehåller visuella resurser kan inkluderas i programpaketet och läsas in vid körning, men inte kod. Om du försöker läsa in en SWF-fil som innehåller ActionScript-kod genereras ett fel. En dialogruta med ett meddelande om ”Okompilerad ActionScript-kod” visas då i programmet.

Se även
Paketera och läsa in flera SWF-filer i AIR-program på iOS (på engelska)

Använda klasserna ProLoader och ProLoaderInfo

Flash Player 9 och senare, Adobe AIR 1.0 och senare samt krav på Flash Professional CS5.5

För att få hjälp med fjärranslutna delade bibliotek (RSL) introduceras klasserna fl.display.ProLoader och fl.display.ProLoaderInfo i Flash Professional CS5.5. Dessa klasser speglar klasserna flash.display.Loader och flash.display.LoaderInfo, men de ger ett mer konsekvent inläsningsbeteende.

Med ProLoader får du speciell hjälp med inläsning av SWF-filer där TLF (Text Layout Framework) med RSL-förinläsning används. Under körningen krävs för SWF-filer som förinläser andra SWF- eller SWZ-filer, till exempel TLF, en SWF-wrapperfil som endast är intern. Den extra komplexiteten som SWF-wrapperfilen innebär kan resultera i oönskade beteenden. ProLoader löser problemet genom att läsa in dessa filer som om de vore vanliga SWF-filer. Den lösningen som används i ProLoader-klassen är generell för användaren och kräver inga speciella åtgärder i ActionScript. Dessutom läser ProLoader in vanligt SWF-innehåll korrekt.

I Flash Professional CS 5.5 och senare är det säkert att ersätta all användning av Loader-klassen med ProLoader-klassen. Exportera sedan programmet till Flash Player 10.2 eller senare så att ProLoader får åtkomst till nödvändiga ActionScript-funktioner. Du kan även använda ProLoader för tidigare versioner av Flash Player som har stöd för ActionScript 3.0. Du får emellertid ut maximalt av ProLoader-funktionerna endast med Flash Player 10.2 eller senare. Använd alltid ProLoader när du använder TLF i Flash Professional CS5.5 eller senare. Det finns inget behov av ProLoader i andra miljöer än Flash Professional.

Viktigt! *För SWF-filer som publiceras i Flash Professional CS5.5 och senare, kan du alltid använda klasserna fl.display.ProLoader och fl.display.ProLoaderInfo i stället för flash.display.Loader och flash.display.LoaderInfo.*

Problem åtgärdade med klassen ProLoader

ProLoader-klassen åtgärdar problem som den äldre Loader-klassen inte kunde hantera. Dessa problem har att göra med RSL-förinläsning av TLF-bibliotek. Det gäller speciellt för SWF-filer som använder ett Loader-objekt för att läsa in andra SWF-filer. Bland annat åtgärdas följande problem:

- **Skripthantering mellan den inläsande och den inlästa filen sker inte så som förväntat.** Klassen ProLoader anger automatiskt den inläsande SWF-filen som överordnad till den inlästa SWF-filen. Kommunikationen sker sålunda från den inläsande SWF-filen direkt till den inlästa SWF-filen.
- **SWF-programmet måste aktivt hantera inläsningsprocessen.** För detta krävs att fler händelser implementeras, till exempel `added`, `removed`, `addedToStage` och `removedFromStage`. Om ditt program är avsett för Flash Player 10.2 eller senare, tar ProLoader bort behovet att detta extraarbete.

Uppdatera kod för att använda ProLoader i stället för Loader

Eftersom ProLoader speglar klassen Loader kan du enkelt växla klasserna i din kod. I följande exempel visas hur befintlig kod uppdateras för att använda den nya klassen:

```
import flash.display.Loader;
import flash.events.Event;
var l:Loader = new Loader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

Den här koden kan uppdateras för att använda ProLoader enligt följande:

```
import fl.display.ProLoader;
import flash.events.Event;
var l:ProLoader = new ProLoader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

Exempel på visningsobjekt: SpriteArranger

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Exempelprogrammet SpriteArranger bygger på exempelprogrammet GeometricShapes, som beskrivs separat i *Learning ActionScript 3.0*.

Exempelprogrammet SpriteArranger demonstrerar flera koncept i hanteringen av visningsobjekt:

- Utöka visningsobjektklasser
- Lägga till objekt i visningslistan
- Lager med visningsobjekt och arbeta med behållare för visningsobjekt
- Besvara visningsobjekthändelser
- Använda egenskaper och metoder för visningsobjekt

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna för SpriteArranger finns i mappen Examples/SpriteArranger. Programmet består av följande filer:

Fil	Beskrivning
SpriteArranger.mxml eller SpriteArranger.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/SpriteArranger/CircleSprite.as	En klass som definierar en typ av Sprite-objekt som återger en cirkel på skärmbilden.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	En klass som definierar arbetsytan, vilket är en behållare som innehåller GeometricSprite-objekt.
com/example/programmingas3/SpriteArranger/SquareSprite.as	En klass som definierar en typ av Sprite-objekt som återger en fyrkant på skärmbilden.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	En klass som definierar en typ av Sprite-objekt som återger en triangel på skärmbilden.
com/example/programmingas3/SpriteArranger/GeometricSprite.as	En klass som utökar Sprite-objektet, vilket används för att definiera en form på skärmbilden. CircleSprite, SquareSprite och TriangleSprite utökar alla den här klassen.
com/example/programmingas3/geometricshapes/IGeometricShape.as	Det grundläggande gränssnittet för metoddefinitioner som implementeras av alla klasser för geometriska former.
com/example/programmingas3/geometricshapes/IPolygon.as	Ett gränssnitt för metoddefinitioner som ska implementeras av klasser för geometriska former med flera sidor.
com/example/programmingas3/geometricshapes/RegularPolygon.as	En typ av geometrisk liksidig form där sidorna är symmetriskt placerade runt formens mittpunkt.
com/example/programmingas3/geometricshapes/Circle.as	En typ av geometrisk form som definierar en cirkel.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	En underklass till RegularPolygon som definierar en liksidig triangel.
com/example/programmingas3/geometricshapes/Square.as	En underklass till RegularPolygon som definierar en liksidig rektangel.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	En klass som innehåller en standardmetod för att skapa former med given formtyp och storlek.

Definiera klasserna SpriteArranger

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med programmet SpriteArranger kan användaren lägga till flera olika visningsobjekt till arbetsytan på skärmen.

Med klassen DrawingCanvas definieras ett ritområde, en typ av behållare för visningsobjekt, där användaren kan lägga till former på skärmbilden. Dessa former på skärmbilden är instanser av en av underklasserna i klassen GeometricSprite.

Klassen DrawingCanvas

I Flex måste alla underordnade visningsobjekt som läggs till i ett Container-objekt vara en klass som härrör från klassen mx.core.UIComponent. Det här programmet lägger till en instans av klassen DrawingCanvas som ett underordnat objekt till ett mx.containers.VBox-objekt, vilket definieras i MXML-koden i filen SpriteArranger.mxml. Arvet definieras i klassen DrawingCanvas' deklaration, på följande vis:

```
public class DrawingCanvas extends UIComponent
```

Klassen UIComponent ärver från klasserna DisplayObject, DisplayObjectContainer och Sprite, och koden i klassen DrawingCanvas använder metoderna och egenskaperna från dessa klasser.

Klassen DrawingCanvas utökar klassen Sprite och detta arv definieras i klassdeklarationen för DrawingCanvas enligt följande:

```
public class DrawingCanvas extends Sprite
```

Klassen Sprite är en underklass till klasserna DisplayObjectContainer och DisplayObject. I klassen DrawingCanvas används metoder och egenskaper från de båda sistnämnda klasserna.

Konstruktormetoden `DrawingCanvas()` anger objektet Rectangle, `bounds`, en egenskap som används senare vid ritning av arbetsytans kontur. Därefter anropas metoden `initCanvas()` enligt följande:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Som visas i följande exempel används metoden `initCanvas()` för att definiera olika egenskaper i objektet DrawingCanvas, som skickades som argument till konstruktorfunktionen:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

Metoden `initCanvas()` anropar sedan metoden `drawBounds()` som ritar arbetsytan med klassen DrawingCanvas' egenskap `graphics`. Egenskapen `graphics` ärvs från klassen Shape.

```
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();
```

Följande övriga metoder i klassen DrawingCanvas anropas baserat på användarinteraktioner med programmet:

- Metoderna `addShape()` och `describeChildren()` som beskrivs i ”Lägga till visningsobjekt till arbetsytan” på sidan 206
- Metoderna `moveToBack()`, `moveDown()`, `moveToFront()` och `moveUp()` som beskrivs i ”Ändra visningsobjektens ordning” på sidan 208
- Metoden `onMouseUp()` som beskrivs i ”Klicka och dra visningsobjekt” på sidan 207

Klassen GeometricSprite och tillhörande underklasser

Varje visningsobjekt som användaren kan lägga till på arbetsytan är en instans av en av följande underklasser för klassen GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

Klassen GeometricSprite utökar klassen flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

Klassen GeometricSprite innehåller ett antal egenskaper som är gemensamma för alla GeometricSprite-objekt. Dessa utgör en uppsättning i konstruktorfunktionen baserat på de parametrar som skickas till funktionen. Till exempel:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

Egenskapen `geometricShape` för klassen GeometricSprite definierar gränssnittet IGeometricShape, vilket definierar formens matematiska egenskaper, men inte de visuella egenskaperna. De klasser som implementerar IGeometricShape-gränssnittet definieras i exempelprogrammet GeometricShapes som beskrivs i *Learning ActionScript 3.0*.

I klassen GeometricSprite definieras metoden `drawShape()`, som förfinas ytterligare i åsidosättningsdefinitionerna i respektive underklass i GeometricSprite. Mer information finns i avsnittet ”Lägga till visningsobjekt till arbetsytan” nedan.

I klassen GeometricSprite ingår även följande metoder:

- Metoderna `onMouseDown()` och `onMouseUp()` som beskrivs i ”Klicka och dra visningsobjekt” på sidan 207
- Metoderna `showSelected()` och `hideSelected()` som beskrivs i ”Klicka och dra visningsobjekt” på sidan 207

Lägga till visningsobjekt till arbetsytan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När användaren klickar på Lägg till form-knappen anropas metoden `addShape()` i klassen DrawingCanvas. En ny GeometricSprite-klass instansieras genom anrop av lämplig konstruktorfunktion i en av GeometricSprite-underklasserna enligt följande exempel:

```
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);
}
```

Varje konstruktormetod anropar metoden `drawShape()` där egenskapen `graphics` används för klassen (ärvs från klassen Sprite) så att lämplig vektorgrafik ritas. Exempelvis innehåller metoden `drawShape()` i klassen CircleSprite följande kod:

```
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);
```

Den näst sista raden i funktionen `addShape()` anger egenskapen `alpha` för visningsobjektet (ärvs från klassen DisplayObject). På så vis blir alla visningsobjekt som läggs till på arbetsytan något genomskinliga och användaren kan se objekten som finns under.

Med den sista raden i metoden `addChild()` läggs det till nya visningsobjektet i listan med underordnade objekt för instansen av klassen DrawingCanvas, som redan finns med i visningslistan. Detta resulterar i att det nya visningsobjektet visas på scenen.

Gränssnittet för programmet innehåller två textfält; `selectedSpriteTxt` och `outputTxt`. Textegenskaperna för dessa textfält uppdateras med information om objekten GeometricSprite som har lagts till på arbetsytan eller valts av användaren. Klassen GeometricSprite hanterar denna informationsrapporterande åtgärd genom att åsidosätta metoden `toString()` enligt följande:

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

Egenskapen `shapeType` anges med lämpligt värde i konstruktormetoden för respektive GeometricSprite-underklass. Exempelvis kan metoden `toString()` returnera följande värde för en CircleSprite-instans som nyligen lagts till i DrawingCanvas-instansen:

```
Circle of size 50 at 0, 0
```

Metoden `describeChildren()` i klassen DrawingCanvas körs som slinga igenom arbetsytans lista med underordnade, där egenskapen `numChildren` (ärvs från klassen DisplayObjectContainer) används som villkor till `for`-slingan. En lista med strängar för alla underordnade objekt genereras enligt följande:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

Den resulterande strängen används för att ange egenskapen `text` i textfältet `outputTxt`.

Klicka och dra visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När användaren klickar på en GeometricSprite-instans anropas händelsehanteraren `onMouseDown()`. Händelsehanteraren anges med avlyssning för mus ned-händelser i konstruktorfunktionen i klassen GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

Metoden `onMouseDown()` anropar sedan metoden `showSelected()` i objektet GeometricSprite. Om det är första gången för objektet som den här metoden anropas skapas det nya Shape-objektet `selectionIndicator` och egenskapen `graphics` i Shape-objektet används för att rita en röd rektangelmarkering:

```
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);
```

Om det inte är första gången som metoden `onMouseDown()` anropas anges egenskapen `visible` för formen `selectionIndicator` (ärvs från klassen DisplayObject):

```
this.selectionIndicator.visible = true;
```

Metoden `hideSelected()` döljer formen `selectionIndicator` för tidigare valda objekt genom att ange egenskapen `visible` till `false`.

Händelsehanterarmetoden `onMouseDown()` anropar även metoden `startDrag()` (ärvs från klassen Sprite), som innehåller följande kod:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

På så vis kan användaren dra det valda objektet över arbetsytan, inom gränserna angivna med rektangeln `boundsRect`.

När användaren släpper musknappen skickas händelsen `mouseUp`. Konstruktormetoden för DrawingCanvas anger följande händelseavlyssnare:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Den här händelseavlyssnaren anges för objektet DrawingCanvas, i stället för separata GeometricSprite-objekt. Detta beror på, att när objektet GeometricSprite dras kan det hamna bakom ett annat visningsobjekt (ett annat GeometricSprite-objekt) när musknappen släpps upp. Visningsobjektet i förgrunden skulle då ta emot den avslutande mus upp-händelsen, motsatt visningsobjektet som användaren drar. Genom att lägga till avlyssnaren till objektet DrawingCanvas säkerställs att händelsen alltid hanteras.

Metoden `onMouseUp()` anropar metoden `onMouseUp()` i objektet GeometricSprite, vilket i sin tur anropar metoden `stopDrag()`.

Ändra visningsobjektens ordning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Användargränssnittet för programmet innehåller knapparna Flytta längst bak (Move Back), Flytta ner (Move Down), Flytta upp (Move Up) och Flytta längst fram (Move to Front). När användaren klickar på någon av knapparna anropas motsvarande metod i klassen DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` eller `moveToFront()`. I till exempel metoden `moveToBack()` ingår följande kod:

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

I metoden används metoden `setChildIndex()` (ärvs från klassen DisplayObjectContainer) för att placera visningsobjektet i indexposition 0 i listan med underordnade objekt för DrawingCanvas-instansen (`this`).

Metoden `moveDown()` fungerar ungefär på samma sätt, förutom att indexpositionen för visningsobjektet minskas med 1 i listan med underordnade objekt för DrawingCanvas-instansen:

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Metoderna `moveUp()` och `moveToFront()` fungerar på liknande sätt som metoderna `moveToBack()` och `moveDown()`.

Kapitel 11: Arbeta med geometri

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Paketet flash.geom innehåller klasser för att definiera geometriska objekt som punkter, rektanglar och omformningsmatriser. Du använder dessa klasser för att definiera objektsegenskaper som används i andra klasser.

Fler hjälpavsnitt

paketet flash.geom

Geometrins grunder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Paketet flash.geom innehåller klasser för att definiera geometriska objekt som punkter, rektanglar och omformningsmatriser. Dessa klasser behöver nödvändigtvis inte ha någon egen funktionalitet, utan används ofta för att definiera objektsegenskaper som används i andra klasser.

Alla geometriklasser baseras på föreställningen att platserna på scenen är representerade som en tvådimensionell yta. Skärmen ses som ett plan med en vågrät (x) och en lodrät (y) axel. Alla platser (eller *punkter*) på scenen kan representeras av ett par x- och y-värden, d.v.s. platsens *koordinater*.

För varje visningsobjekt, inklusive scenen, finns en separat *koordinatmodell*. Koordinatmodellen är objektets egen graf för att rita ned platserna för underordnade visningsobjekt, ritningar och så vidare. *Origo* är vid koordinaterna 0, 0 (där x- och y-axeln möts) och sitter i det övre vänstra hörnet av visningsobjektet. Platsen för origo är alltid densamma för scenen, men är inte alltid densamma för andra visningsobjekt. Värden på x-axeln blir högre från vänster till höger och lägre från höger till vänster. Till vänster om origopunkten är x-koordinaten negativ. På y-axeln blir koordinatvärden för Flash-körningen däremot högre uppifrån och ner och lägre nerifrån och upp, till skillnad från traditionella koordinatsystem där förhållandet är det omvända. Värden ovanför origopunkten är negativa y-koordinater. Eftersom origopunkten för scenens koordinatmodell är i det övre vänstra hörnet av scenen har de flesta objekten på scenen en x-koordinat som är högre än 0 och lägre än scenens bredd. Samma objekt har en y-koordinat som är högre än 0 och lägre än scenens höjd.

Du använder instanser av klassen Point för att representera enskilda punkter i en koordinatmodell. Du kan skapa en Rectangle-instans för att representera ett rektangulärt område i en koordinatmodell. Erfarna användare kan använda en Matrix-instans för att tillämpa flera eller komplexa omformningar för ett visningsobjekt. Många enkla omformningar, till exempel ändringar av rotation, position och skala, kan tillämpas direkt på ett visningsobjekt med hjälp av objektets egenskaper. Mer information om att tillämpa omformningar med hjälp av visningsobjektsegenskaper finns i ”Ändra visningsobjekt” på sidan 173.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga geometritermer:

Kartesiska koordinater Koordinaterna skrivs vanligtvis som talpar (till exempel 5, 12 eller 17, -23). De båda talen representerar x- respektive y-koordinaten.

Koordinatmodell Det koordinatdiagram i ett visningsobjekt på vilket de underordnade elementen placeras.

Origo Punkten i ett koordinatsystem där x-axeln möter y-axeln. Punkten har koordinaterna 0, 0.

Punkt En plats i en koordinatmodell. I det tvådimensionella koordinatsystemet som används i ActionScript, definieras punkten av platsen längs x-axeln och y-axeln (punktens koordinater).

Registreringspunkt I ett visningsobjekt är det origo (koordinaterna 0, 0) för koordinatmodellen.

Skala Storleken på ett objekt i förhållande till dess ursprungliga storlek. Att skala ett objekt innebär att ändra dess storlek genom att sträcka ut eller förminska det.

Översätta Att ändra en punkts koordinater från ett koordinatsystem till ett annat.

Omformning En justering av en visuell egenskap i en grafik, till exempel rotering av objektet, skaländring, skevning eller förvrängning av formen, eller ändring av färgen.

X-axel Den vågräta axeln i det tvådimensionella koordinatsystem som används i ActionScript.

Y-axel Den lodräta axeln i det tvådimensionella koordinatsystem som används i ActionScript.

Använda Point-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett Point-objekt definierar ett kartesiskt koordinatpar. Det representerar en plats i ett tvådimensionellt koordinatsystem där *x* representerar den vågräta axeln och *y* den lodräta.

Du definierar ett Point-objekt genom att ange `x`- och `y`-egenskaperna enligt följande:

```
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;
```

Hitta rätt avstånd mellan två punkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder metoden `distance()` i klassen Point för att finna rätt avstånd mellan två punkter i en koordinatmodell. I nästa exempel hittas avståndet mellan registreringspunkterna för två objekt, `circle1` och `circle2`, i samma visningsobjektsbehållare:

```
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);
```

Översättning av koordinatmodeller

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om det finns två visningsobjekt i olika visningsobjektsbehållare kan de finnas i olika koordinatmodeller. Du använder metoden `localToGlobal()` i klassen DisplayObject för att översätta koordinaterna till samma (globala) koordinatmodell som på scenen. I nästa exempel hittas avståndet mellan registreringspunkterna för två objekt, `circle1` och `circle2`, i olika visningsobjektsbehållare:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Du kan på liknande sätt hitta avståndet till registreringspunkten för ett visningsobjekt med namnet `target` från en viss punkt på scenen genom att använda metoden `localToGlobal()` i klassen DisplayObject:

```
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);
```

Flytta ett visningsobjekt ett visst antal grader och en viss sträcka

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du använder metoden `polar()` i klassen Point kan du flytta ett visningsobjekt en viss sträcka i en viss vinkel. I nästa exempel flyttas objektet `myDisplayObject` 100 pixlar i 60 grader:

```
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;
```

Andra användningsområden för klassen Point

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda Point-objekt med följande metoder och egenskaper:

Klass	Metoder eller egenskaper	Beskrivning
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()`	Används för att returnera en lista med objekt under en punkt i en visningsobjektsbehållare.
BitmapData	`hitTest()`	Används för att definiera pixlar i BitmapData-objektet samt punkten som du kontrollerar för händelsen.

Klass	Metoder eller egenskaper	Beskrivning

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Används för att definiera positionen för de rektanglar som definierar operationen.

Matris	`deltaTransformPoint()` `transformPoint()`	Används för att definiera punkter för vilka du vill tillämpa en omformning.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		Används för att definiera dessa egenskaper.

Använda Rectangle-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med Rectangle definierar du ett rektangulärt område. Ett Rectangle-objekt har en plats som definieras av *x*- och *y*-koordinaterna för dess övre vänstra hörn och egenskaper för `width` och `height`. Du kan definiera dessa egenskaper för ett nytt Rectangle-objekt genom att anropa konstruktorn `Rectangle()` enligt följande:

```
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);
```

Ändra storlek på och flytta Rectangle-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns olika sätt att välja mellan när du vill flytta och ändra storlek på Rectangle-objekt.

Du kan flytta Rectangle-objektet direkt genom att ändra dess `x`- och `y`-egenskaper. Den här ändringen har ingen effekt på Rectangle-objektets bredd eller höjd.

```
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)
```

När du ändrar egenskapen `left` eller `top` för ett Rectangle-objekt omplaceras rektangeln, vilket följande kod visar. Rektangelns `x`- och `y`-egenskaper matchar egenskaperna `left` respektive `top`. Platsen för Rectangle-objektets nedre vänstra hörn ändras däremot inte, utan i stället ändras storleken.

```
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=20, y=30, w=80, h=20)
```

På samma sätt ändras inte platsen för det övre vänstra hörnet om du ändrar egenskapen `bottom` eller `right` för ett Rectangle-objekt, som nästa exempel visar. Rektangelns storlek ändras:

```
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)
```

Du kan också flytta ett Rectangle-objekt med hjälp av metoden `offset()`. Gör så här:

```
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)
```

Metoden `offsetPt()` fungerar ungefär på samma sätt, förutom att den tar ett Point-objekt som sin parameter, i stället för förskjutningsvärdena *x* och *y*.

Med metoden `inflate()` kan du också ändra storleken på ett Rectangle-objekt. Metoden innehåller två parametrar, `dx` och `dy`. Parametern `dx` representerar det antal pixlar som rektangelns vänstra och högra sidor flyttas från centrum. Parametern `dy` representerar det antal pixlar som rektangelns övre och nedre sidor flyttas från centrum:

```
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)
```

Metoden `inflatePt()` fungerar ungefär på samma sätt, förutom att den tar ett Point-objekt som sin parameter, i stället för värdena `dx` och `dy`.

Hitta föreningar och överlappande områden i Rectangle-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder metoden `union()` för att hitta det rektangulära området som bildas av de båda rektanglarnas gränser:

```
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)
```

Du använder metoden `intersection()` för att hitta det rektangulära området som bildas av de båda rektanglarnas överlappande områden:

```
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)
```

Om du vill ta reda på om två rektanglar överlappar varandra använder du metoden `intersects()`. Du kan också använda metoden `intersects()` för att ta reda på om ett visningsobjekt finns i ett visst område på scenen. I följande kodexempel antas att koordinatmodellen för visningsobjektsbehållaren med objektet `circle` är densamma som för scenen. I exemplet visas hur du använder metoden `intersects()` för att bestämma om ett visningsobjekt, `circle`, överlappar angivna områden på scenen så som de definieras av Rectangle-objekten `target1` och `target2`:

```
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
```

På liknade sätt kan du använda metoden `intersects()` för att ta reda på om markeringsramen för de båda visningsobjekten överlappar varandra. Med metoden `getRect()` för DisplayObject-klassen inkluderar du eventuellt utrymme som linjerna för ett visningsobjekt kan lägga till i ett gränsområde.

Andra användningsområden för Rectangle-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Rectangle-objekt används i följande metoder och egenskaper:

Klass	Metoder eller egenskaper	Beskrivning
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `drawWithQuality()`, `encode()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` och `threshold()`	Används som typ för vissa parametrar för att definiera ett område av BitmapData-objektet.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Används som datatyp för egenskapen eller den returnerade datatypen.
PrintJob	`addPage()`	Används för att definiera parametern `printArea`.
Sprite	`startDrag()`	Används för att definiera parametern `bounds`.
TextField	`getCharBoundaries()`	Används som returnerad värdetyp.
Transform	`pixelBounds`	Används som datatyp.

Använda Matrix-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Matrix representerar en omformningsmatris som avgör hur punkter från en koordinatmodell ska kopplas till en annan. Du kan utföra olika grafiska omformningar på ett visningsobjekt genom att ställa in ett Matrix-objekts egenskaper och använda Matrix-objektet på ett Transform-objekts `matrix`-egenskap, och sedan använda Transform-objektet som visningsobjektets `transform`-egenskap. Dessa omformningsfunktioner är bland annat translation (omplacering av *x* och *y*), rotation, skalning och skevning.

Det är enklare att använda metoden `createBox()` för att definiera en matris trots att det går genom att ändra egenskaperna (`a`, `b`, `c`, `d`, `tx`, `ty`) i ett Matrix-objekt. Den här metoden innehåller parametrar för att direkt definiera skalning, rotation och omformningseffekter för den resulterande matrisen. I följande exempel skapas ett Matrix-objekt som ändrar skalan på ett objekt vågrätt med 2,0 och lodrätt med 3,0, roterar det 45 grader samt flyttar (omformar) det 10 pixlar åt höger och 20 pixlar nedåt:

```
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);
```

Du kan också ändra skala, rotation och omformningseffekter för Matrix-objekt med metoderna `scale()`, `rotate()` och `translate()`. Observera att dessa metoder kombinerar värdena för det befintliga Matrix-objektet. I nästa exempel anges ett Matrix-objekt som ändrar objektets skala med faktorn 4 och roterar det 60 grader genom att metoderna `scale()` och `rotate()` anropas två gånger:

```
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;
```

Om du vill skevningsomforma ett Matrix-objekt ska du ändra egenskapen `b` eller `c`. Genom ändring av egenskapen `b` skevar matrisen lodrätt medan ändring av egenskapen `c` får den att skeva vågrätt. I nästa exempel skevar Matrix-objektet `myMatrix` lodrätt med en faktor på 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Du kan tillämpa en matrisomformning på egenskapen `transform` för ett visningsobjekt. I nästa exempel tillämpas en matrisomformning på ett visningsobjekt med namnet `myDisplayObject`:

```
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;
```

I den första raden ställs ett matrisobjekt in för den aktuella matrisomformningen som används i visningsobjektet `myDisplayObject` (egenskapen `matrix` i egenskapen `transformation` i visningsobjektet `myDisplayObject`). På det här sättet får de anropade metoderna för Matrix-klassen en kumulativ effekt på visningsobjektets befintliga plats, skala och rotation.

Obs! *Klassen ColorTransform ingår också i flash.geometry-paketet. Den här klassen används för att ställa in egenskapen `colorTransform` för ett Transform-objekt. Eftersom ingen geometrisk omformning sker beskrivs den inte närmare här. Mer information finns i beskrivningen för klassen ColorTransform i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.*

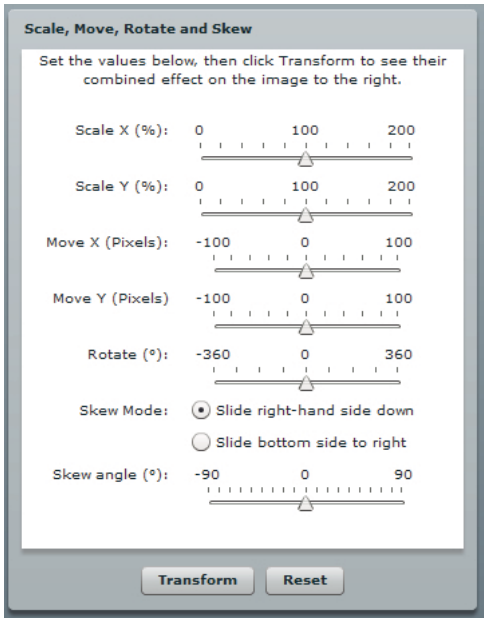
Exempel på geometri: tillämpa en matrisomformning på ett visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I programmet DisplayObjectTransformer visas olika sätt att använda klassen Matrix för att omforma ett visningsobjekt, inklusive följande:

- Rotering av visningsobjekt
- Skalning av visningsobjekt
- Översättning (flyttning) av visningsobjekt
- Skevning av visningsobjekt

I programmet finns ett gränssnitt för att justera parametrarna för matrisinformationen enligt följande:



När användaren klickar på knappen Transform kommer aktuella omformningsinställningar att tillämpas.

Det ursprungliga visningsobjektet och visningsobjektet som har roterats -45° och skalförändrats 50 %

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Du hittar programfiler för DisplayObjectTransformer i mappen Samples/DisplayObjectTransformer. Programmet består av följande filer:

Fil	Beskrivning
DisplayObjectTransformer.mxml eller DisplayObjectTransformer.fla	Huvudprogramfilen i Flash (FLA) eller Flex (MXML)
com/example/programmingas3/geometry/MatrixTransformer.as	En klass som innehåller metoder för att tillämpa matrisomformningar.
img/	En katalog med bildfiler som används i programmet.

Definiera klassen MatrixTransformer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen MatrixTransformer innehåller statiska metoder för att tillämpa geometriska omformningar på matrisobjekt.

Metoden transform()

Metoden `transform()` innehåller följande parametrar:

- `sourceMatrix` – Indatamatrisen som metoden omformar
- `xScale` och `yScale` – Skalfaktorn *x* och *y*
- `dx` och `dy` – Översättningsmängden *x* och *y*, i pixlar
- `rotation` – Rotationen i grader
- `skew` – Skevningsfaktorn i procent
- `skewType` – Riktningen på skevningen, antingen `"right"` eller `"left"`

Det returnerade värdet är den resulterande matrisen.

Med metoden `transform()` anropas följande statiska metoder för klassen:

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

Alla returnerar källmatrisen med den tillämpade omformningen.

Metoden skew()

Metoden `skew()` skevar matrisen genom att ändra dess egenskaper `b` och `c`. En valfri parameter, `unit`, används för att bestämma vilken enhet som ska användas för att definiera skevningsvinkeln. Om nödvändigt konverteras `angle`-värdet för metoden till radianer:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Ett `skewMatrix`-matrisobjekt skapas och ändras för att tillämpa skevningsomformningen. Till att börja med har identitetsmatrisen följande utseende:

```
var skewMatrix:Matrix = new Matrix();
```

Parametrarna `skewSide` avgör vilken sida som skevningen ska tillämpas på. Om det är `"right"` används följande kod för att ställa in matrisegenskapen `b`:

```
skewMatrix.b = Math.tan(angle);
```

I annat fall är det undersidan som skevas genom att matrisegenskapen `c` ändras enligt följande:

```
skewMatrix.c = Math.tan(angle);
```

Den resulterande skevningen tillämpas sedan på den befintliga matrisen genom att två matriser slås samman enligt följande:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

Metoden scale()

Följande exempel visar hur metoden `scale()` först justerar skalfaktorn om den anges i procent och sedan använder metoden `scale()` för matrisobjektet:

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

Metoden translate()

Metoden `translate()` tillämpar översättningsfaktorerna `dx` och `dy` genom att anropa metoden `translate()` för matrisobjektet enligt följande:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

Metoden rotate()

Metoden `rotate()` används för att konvertera indatarotationsfaktorn till radianer (om den anges i grader eller övertoningar) och därefter anropas metoden `rotate()` för matrisobjektet:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Anropa metoden MatrixTransformer.transform() från programmet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet innehåller ett användargränssnitt för att hämta omformningsparametrar från användaren. Därefter skickas dessa värden tillsammans med egenskapen `matrix` för egenskapen `transform` för visningsobjektet till metoden `Matrix.transform()` enligt följande:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

Programmet tillämpar sedan det returnerade värdet på egenskapen `matrix` för egenskapen `transform` i visningsobjektet vilket därmed aktiverar omformningen:

```
img.content.transform.matrix = tempMatrix;
```

Kapitel 12: Använda Drawing API (teckningsgränssnittet)

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Importerade bilder är viktiga, men funktionen Drawing API (teckningsgränssnittet) som gör att du kan rita linjer och former i ActionScript, ger dig friheten att starta ett program med en tom yta och skapa vilka bilder du vill. Att kunna skapa egen grafik ger stora möjligheter för dina program. Med teknikerna som beskrivs här kan du bland annat skapa ett ritprogram, skapa animerad interaktiv grafik eller programmera egna element för användargränssnittet.

Fler hjälpavsnitt

flash.display.Graphics

Grunderna i Drawing API

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Drawing API är namnet på den inbyggda ActionScript-funktionen som gör att du kan skapa vektorgrafik, d.v.s. linjer, kurvor, former, fyllningar och övertoningar, och visa dem på skärmen med ActionScript. Funktionen finns i klassen flash.display.Graphics. Du kan rita med ActionScript på valfri Shape-, Sprite- eller MovieClip-instans med egenskapen `graphics` som definieras i varje klass. (Alla dessa klassers `graphics`-egenskap är en instans av klassen Graphics.)

Om du precis har börjat rita med kod finns det flera metoder i klassen Graphics som gör det enkelt att rita vanliga former som cirklar, ellipser, rektanglar och rektanglar med rundade hörn. Du kan rita dem som tomma linjer eller fyllda former. Om du behöver mer avancerade funktioner innehåller klassen Graphics även metoder för att rita linjer och kvadratiska bezierkurvor, som du kan använda tillsammans med de trigonometriska funktionerna i klassen Math för att skapa de former du behöver.

Flash-körningar (till exempel Flash Player 10 och Adobe AIR 1.5 eller senare versioner) lägger till ytterligare ett rit-API som du kan använda för att skriva kod som ritar hela former med ett enda kommando. När du har lärt dig hur klassen Graphics och uppgifterna som beskrivs i ”Grunderna i Drawing API” fungerar, fortsätter du med ”Använda avancerat Drawing API” på sidan 235 där du får lära dig mer om funktionerna i Drawing API.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kan stöta på när du använder Drawing API:

Ankarpunkt En av de två slutpunkterna i en kvadratisk bezierkurva.

Kontrollpunkt Den punkt som definierar riktningen och kurvningsmängden i en kvadratisk bezierkurva. Den kurvade linjen når aldrig kontrollpunkten, men linjen kurvas som om den hade ritats mot kontrollpunkten.

Koordinatmodell Det koordinatdiagram i ett visningsobjekt på vilket de underordnade elementen placeras.

Fyllning Den täckande inre delen av en form som har en linje ifylld med färg, eller en hel form utan kontur.

Övertoning En färg som består av en gradvis övergång från en färg till en eller flera andra färger (till skillnad från en heltäckande färg).

Punkt En plats i en koordinatmodell. I det tvådimensionella koordinatsystemet som används i ActionScript, definieras en punkt genom sin position i förhållande till x- och y-axeln (punktens koordinater).

Kvadratisk bezierkurva En typ av kurva som definieras av en viss matematisk formel. I den här typen av kurva beräknas kurvans form utifrån ankarpunkternas (kurvans slutpunkter) positioner och av en kontrollpunkt som definierar kurvans storlek och riktning.

Skala Storleken på ett objekt i förhållande till dess ursprungliga storlek. Att skala ett objekt innebär att ändra dess storlek genom att sträcka ut eller förminska det.

Linje En forms konturdel med en linje ifylld med färg, eller linjerna i en form som inte är ifylld.

Översätta Att ändra en punkts koordinater från ett koordinatsystem till ett annat.

X-axel Den vågräta axeln i det tvådimensionella koordinatsystem som används i ActionScript.

Y-axel Den lodräta axeln i det tvådimensionella koordinatsystem som används i ActionScript.

Klassen Graphics

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla Shape-, Sprite- och MovieClip-objekt har en `graphics`-egenskap som är en instans av klassen Graphics. Klassen Graphics innehåller egenskaper och metoder som används för att rita linjer, fyllningar och former. Om du vill att ett visningsobjekt bara ska användas som en yta att rita innehåll på kan du använda en Shape-instans. En Shape-instans fungerar bättre än andra visningsobjekt för ritning eftersom den inte har de extra funktioner som tar upp plats i klasserna Sprite och MovieClip. Om du vill ha ett visningsobjekt som du kan rita grafiskt innehåll i och dessutom vill att objektet ska innehålla andra visningsobjekt kan du använda en Sprite-instans. Mer information om hur du avgör vilket visningsobjekt som ska användas för olika uppgifter finns i ”Välja underklassen DisplayObject” på sidan 172.

Rita linjer och kurvor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

All ritning som görs med en Graphics-instans baseras på grundläggande ritning med linjer och kurvor. Därför måste all ActionScript-ritning göras i samma steg:

- Definiera linje- och fyllningsformat.
- Ställ in ursprungligt ritläge.
- Rita linjer, kurvor och former (eventuellt flytta ritpunkten).
- Avsluta med att skapa en fyllning om det behövs.

Definiera linje- och fyllningsformat

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att kunna rita med egenskapen `graphics` i en Shape-, Sprite- eller MovieClip-instans måste du först definiera formatet (linjens storlek och färg samt fyllningens färg) som ska användas. Precis som när du använder ritverktygen i Adobe® Flash® Professional eller något annat ritprogram kan du rita med eller utan ramlinje och med eller utan fyllningsfärg när du ritar med ActionScript. Du anger ramlinjens storlek med metoden `lineStyle()` eller `lineGradientStyle()`. Om du vill skapa en heldragen linje använder du metoden `lineStyle()`. När du anropar den här metoden är de värden du oftast anger de tre första parametrarna: linjens tjocklek, färg och alfa. Den här kodraden gör till exempel att formen `myShape` ritar linjer två pixlar tjocka, röda (0x990000) och 75 % ogenomskinliga:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

Standardvärdet för parametern alpha är 1.0 (100 %), så du kan utelämna parametern om du vill ha en helt ogenomskinlig linje. Metoden `lineStyle()` tar även två ytterligare parametrar för pixelutjämning och skalningsläge. Mer information om hur du använder dessa parametrar finns i beskrivningen av metoden `Graphics.lineStyle()` i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Om du vill skapa en övertoningslinje använder du metoden `lineGradientStyle()`. Den här metoden beskrivs i ”Skapa övertoningslinjer och fyllningar” på sidan 227.

Om du vill skapa en fylld form anropar du metoden `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` eller `beginShaderFill()` innan du startar ritningen. Den mest grundläggande av dessa är metoden `beginFill()`. Den tar två parametrar: fyllningsfärgen och (eventuellt) ett alfavärde för fyllningsfärgen. Om du till exempel vill rita en form med heltäckande grön fyllning använder du följande kod (förutsatt att du ritar på ett objekt som heter `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

När en fyllningsmetod anropas avslutas automatiskt en eventuell tidigare fyllning innan den nya startas. Om en metod med som anger ett linjeformat anropas ersätts den tidigare linjen, men den tidigare angivna fyllningen ändras inte, och vice versa.

När du har angett linjeformat och fyllningsegenskaper är nästa steg att ange startpunkt för teckningen. Graphics-instansen har en ritpunkt, som spetsen på en penna på ett papper. Oavsett var ritpunkten finns är det där nästa ritåtgärd börjar. Från början har Graphics-objektet ritpunkten vid punkten 0, 0 i koordinatsystemet för det objekt som det ritar på. Om du vill börja rita på ett annat ställe anropar du metoden `moveTo()` innan du anropar någon av ritmetoderna. Detta motsvarar att lyfta pennan från papperet och flytta den till en ny position.

När ritpunkten är på plats ritar du med hjälp av en serie anrop till ritmetoderna `lineTo()` (räta linjer) och `curveTo()` (kurvade linjer).

När du ritar kan du anropa metoden `moveTo()` när som helst om du vill flytta ritpunkten till en ny position utan att rita.

När du ritar kan du stänga fyllningen genom att anropa metoden `endFill()` om du har angett en fyllningsfärg. Om du inte har ritat en sluten form (d.v.s. om ritpunkten inte är vid formens startpunkt när du anropar `endFill()`) stänger Flash-körningen automatiskt formen när du anropar `endFill()` genom att rita en rät linje från den aktuella ritpunkten till den plats som angavs i det senaste `moveTo()`-anropet. Om du har startat en fyllning men inte har anropat `endFill()` stänger `beginFill()` (eller någon av de andra fyllningsmetoderna) den aktiva fyllningen och startar en ny.

Rita raka linjer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du anropar metoden `lineTo()` ritar Graphics-objektet en rak linje från den nuvarande ritpunkten till de koordinater du anger som de två parametrarna i metodanropet med det linjeformat du har angett. Den här kodraden placerar till exempel ritpunkten vid punkten 100, 100 och ritar en linje till punkten 200, 200:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

I följande exempel ritas en röd och grön trianglar som är 100 pixlar hög:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
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);

// green triangle, starting at point 200, 0
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);
```

Rita kurvor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `curveTo()` är en kvadratisk bezierkurva. Den ritar en båge som ansluter två punkter (kallas ankarpunkter) och böjer sig mot en tredje punkt (kallas kontrollpunkten). I Graphics-objektet används den nuvarande ritpositionen som första ankarpunkt. När du anropar metoden `curveTo()` anger du fyra parametrar: kontrollpunktens x- och y-koordinater följt av den andra ankarpunktens x- och y- koordinater. Följande kod ritar till exempel en kurva från punkten 100, 100 till 200, 200. Eftersom kontrollpunkten är 175, 125 skapas en kurva som rör sig till höger och sedan nedåt.

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

I följande exempel ritas röda och gröna cirkulära objekt med bredden och höjden 100 pixlar. Observera att den kvadratiska bezierekvationen gör att dessa inte är perfekta cirklar:

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
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);

// green circular shape
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);
```

Rita former med inbyggda metoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För enkelhetens skull har ActionScript 3.0 metoder som ritar vanliga former som cirklar, ellipser, rektanglar och rektanglar med rundade hörn åt dig. Dessa är metoderna `drawCircle()`, `drawEllipse()`, `drawRect()` och `drawRoundRect()` i klassen Graphics. Dessa metoder kan användas i stället för metoderna `lineTo()` och `CurveTo()`. Observera dock att du ändå måste ange linje- och fyllningsformat innan du anropar metoderna.

I följande exempel återskapas exemplet med röda, gröna och blå kvadrater med bredden och höjden 100 pixlar. I koden används metoden `drawRect()`. Dessutom anges att fyllningsfärgen ska ha alfavärdet 50 % (0.5):

```
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);
```

I ett Sprite- eller MovieClip-objekt visas alltid ritinnehållet som skapas med `graphics`-egenskapen bakom alla underordnade objekt som finns i objektet. Dessutom är inte `graphics`-objektets innehåll ett separat visningsobjekt. Därför visas det inte i listan över ett Sprite- eller MovieClip-objekts underordnade objekt. I följande Sprite-objekt ritas till exempel en cirkel med `graphics`-egenskapen och ett TextField-objekt finns i listan med underordnade visningsobjekt:

```
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);
```

Observera att TextField-objektet visas ovanpå cirkeln som visas med graphics-objektet.

Skapa övertoningslinjer och fyllningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Graphics-objektet kan även rita ramlinjer och fyllningar med övertoningar i stället för heltäckande färger. En övertoningslinje skapas med metoden `lineGradientStyle()` och en övertoningsfyllning skapas med metoden `beginGradientFill()`.

Båda metoderna tar samma parametrar. De första fyra är obligatoriska: type, colors, alphas och ratios. De resterande fyra är valfria och används för avancerad anpassning.

- Den första parametern anger vilken typ av övertoning du skapar. Godtagbara värden är `GradientType.LINEAR` eller `GradientType.RADIAL`.
- Den andra parametern anger den array med färgvärden som ska användas. I en linjär övertoning ordnas färgerna från vänster till höger. I en radiell övertoning ordnas färgerna inifrån och ut. Färgernas ordning i arrayen representerar den ordning som färgerna ritas i övertoningen.
- Den tredje parametern anger alfagenomskinlighetsvärden för de motsvarande färgerna i den föregående parametern.
- Den fjärde parametern anger förhållanden, eller hur stor vikt varje färg har i övertoningen. Giltiga värden är från 0 till 255. Värdena representerar inte bredd eller höjd utan placering i övertoningen. 0 är övertoningens början, 255 är övertoningens slut. Arrayen med förhållanden måste öka sekventiellt och ha samma antal värden som både färg- och alfaarrayerna som anges av den andra och tredje parametern.

Den femte parametern, omformningsmatrisen, är valfri, men den används ofta eftersom den är ett lätt och bra sätt att styra övertoningens utseende. Parametern tar en Matrix-instans. Det lättaste sättet att skapa ett Matrix-objekt för en övertoning är att använda metoden `createGradientBox()` i klassen Matrix.

Definiera ett Matrix-objekt att använda i en övertoning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoderna `beginGradientFill()` och `lineGradientStyle()` i klassen flash.display.Graphics används för att definiera övertoningar som ska användas i former. När du definierar en övertoning anger du en matris som en av parametrarna för dessa metoder.

Det lättaste sättet att definiera matrisen är att använda metoden `createGradientBox()` i klassen Matrix. Den skapar en matris som används för att definiera övertoningen. Övertoningens skala, rotation och position definieras med parametrarna som skickas till metoden `createGradientBox()`. Metoden `createGradientBox()` tar följande parametrar:

- Övertoningsrutans bredd: den bredd i pixlar som övertoningen ska spridas till.
- Övertoningsrutans höjd: den höjd (i pixlar) som övertoningen ska spridas till.
- Övertoningsrutans rotation: den rotation (i radianer) som ska tillämpas på övertoningen.
- Vågrät translatering: hur långt (i pixlar) som övertoningen ska flyttas vågrätt.
- Lodrät translatering: hur långt (i pixlar) som övertoningen ska flyttas lodrätt.

Ta till exempel en övertoning med följande egenskaper:

- `GradientType.LINEAR`
- Två färger, grönt och blått, med arrayen `ratios` inställd på `[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

I följande exempel visas övertoningar där parametern `rotation` i metoden `createGradientBox()` är olika medan alla andra inställningar är desamma:

`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;`	

I följande exempel visas effekterna på en linjär övertoning från grönt till blått där parametrarna `rotation`, `tx` och `ty` i metoden `createGradientBox()` är olika medan alla andra inställningar är desamma:

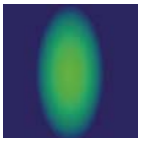
`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`	

`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;`	

Parametrarna `width`, `height`, `tx` och `ty` i metoden `createGradientBox()` påverkar även storleken och positionen för en *radiell* övertoningsfyllning, vilket visas i följande exempel:

`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 25;`
`ty = 0;`
	

Följande kod ger den sista radiella övertoningen som visas:

```
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);
```

Observera att övertoningsfyllningens bredd och höjd avgörs av övertoningsmatrisens bredd och höjd, inte av den bredd och höjd som ritas med Graphics-objektet. När du ritar med Graphics-objektet ritar du det som finns på de koordinaterna i övertoningsmatrisen. Även om du använder en av formmetoderna i ett Graphics-objekt som `drawRect()` sträcks inte övertoningen ut till samma storlek som formen som ritas. Övertoningens storlek måste anges i själva övertoningsmatrisen.

Nedan visas den visuella skillnaden mellan övertoningsmatrisens mått och ritåtgärdens mått:

```
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);
```

I den här koden ritas tre övertoningar med samma fyllningsformat, som anges med en jämn fördelning av rött, grönt och blått. Övertoningarna ritas med metoden `drawRect()` med pixelbredderna 50, 100 respektive 150. Övertoningsmatrisen som anges i metoden `beginGradientFill()` skapas med bredden 100 pixlar. Det innebär att den första övertoningen bara omfattar hälften av övertoningsspektrat, den andra omfattar hela och den tredje omfattar hela och har ytterligare 50 pixlar som är helt blå till höger.

Metoden `lineGradientStyle()` fungerar på ungefär samma sätt som `beginGradientFill()`, förutom att du (utöver att definiera övertoningen) måste ange tjocklek för ramlinjen med `lineStyle()` innan du börjar rita. I följande kod ritas en ruta med en röd, grön och blå övertoningslinje:

```
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);
```

Mer information om klassen Matrix finns i ”Använda Matrix-objekt” på sidan 216.

Använda klassen Math med ritmetoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett Graphics-objekt ritar cirklar och fyrkanter, men det kan också rita mer komplicerade former, särskilt om ritmetoderna används i kombination med egenskaperna och metoderna i klassen Math. Klassen Math innehåller vanliga matematiska konstanter, till exempel `Math.PI` (approximativt 3,14159265...), en konstant för förhållandet mellan en cirkels omkrets och dess diameter. Dessutom innehåller den metoder för trigonometriska funktioner, bland annat `Math.sin()`, `Math.cos()` och `Math.tan()`. Att rita former med dessa metoder och konstanter ger mer dynamiska visuella effekter, särskilt när de används med repetition eller rekursion.

Många metoder i klassen Math kräver cirkulära mått i radianer i stället för i grader. Att konvertera mått mellan dessa enheter är en vanlig användning av klassen Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

I följande exempel skapas en sinusvåg och en cosinusvåg som visar skillnaden mellan metoderna `Math.sin()` och `Math.cos()` för ett visst värde.

```
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);
}
```

Animera med Drawing API

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En fördel med att skapa innehåll med Drawing API är att du inte är begränsad till att placera innehållet en gång. Det du ritar kan ändras genom att behålla och ändra variablerna som används för att rita. Du kan skapa animering genom att ändra variabler och rita om, antingen över en bildpunktsperiod eller med tidsinställning.

I följande kod ändras till exempel visningen i varje bildruta (genom att händelsen `Event.ENTER_FRAME` avlyssnas) genom att gradantalet ökas och Graphics-objektet rensas och ritas om med uppdaterad position.

```
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;
}
```

För att kunna skapa ett resultat som är betydligt annorlunda kan du experimentera med att ändra de ursprungliga startvariablerna i början av koden, `currentDegrees`, `radius` och `satelliteRadius`. Pröva till exempel med att krympa variabeln radius och/eller öka variabeln totalSatellites. Det här är bara ett exempel på hur Drawing API kan användas för att skapa visuellt innehåll vars komplexitet döljer hur enkelt det var att skapa.

Exempel på Drawing API: algoritmisk visuell generator

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I exemplet Algoritmisk visuell generator ritas flera ”satelliter”, eller cirklar, dynamiskt på scenen. Satelliterna rör sig i en cirkulär omloppsbana. Bland annat utforskas följande funktioner:

- använda Drawing API för att rita en grundläggande form med dynamiskt utseende
- koppla användarinteraktivitet till egenskaperna som används i en ritåtgärd
- skapa animering genom att rensa scenen och rita om i varje bildruta.

I exemplet i det föregående delavsnittet animeras en ensam ”satellit” med händelsen `Event.ENTER_FRAME`. Det här exemplet utökar exemplet i det föregående delavsnittet genom att en kontrollpanel med en serie skjutreglage som omedelbart uppdaterade visningen av flera satelliter lades till. I det här exemplet formaliseras koden till externa klasser och koden där satelliter skapas placeras i en slinga. En referens till varje satellit sparas i arrayen `satellites`.

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna finns i mappen Samples/AlgorithmicVisualGenerator. I den mappen finns följande filer:

Fil	Beskrivning
AlgorithmicVisualGenerator.fla	Huvudprogramfilen i Flash Professional (FLA).
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	Den klass som ger programmet dess huvudsakliga funktionalitet, inklusive att rita satelliter på scenen och att svara på händelser från kontrollpanelen för att uppdatera variablerna som påverkar hur satelliterna ritas.
com/example/programmingas3/algorithmic/ControlPanel.as	En klass som hanterar användarinteraktivitet med flera skjutreglage och skickar händelser när interaktivitet sker.
com/example/programmingas3/algorithmic/Satellite.as	En klass som representerar visningsobjektet som kretsar i en omloppsbana runt en mittpunkt och som innehåller egenskaper som relaterar till det aktuella ritningsläget.

Ställa in avlyssnarna

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet skapar först tre avlyssnare. Den första lyssnar efter en händelse som skickas från kontrollpanelen om att satelliterna behöver ritas om. Den andra lyssnar efter ändringar av SWF-filens scenstorlek. Den tredje lyssnar efter varje bildruta som går i SWF-filen och används för att rita om med funktionen `doEveryFrame()`.

Skapa satelliterna

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När dessa avlyssnare har ställts in anropas funktionen `build()`. Funktionen anropar först funktionen `clear()`, som tömmer arrayen `satellites` och rensar allt som tidigare ritats på scenen. Detta är nödvändigt eftersom funktionen `build()` kan anropas igen varje gång kontrollpanelen skickar en händelse om att det ska göras, till exempel när färginställningarna har ändrats. I så fall måste satelliterna tas bort och skapas om.

Funktionen skapar sedan satelliterna och ställer in de startegenskaper som behövs för att skapa dem. Det är bland annat variabeln `position` som startar på en slumpmässig position i omloppsbanan, och variabeln `color`, som i det här exemplet inte ändras efter att satelliten har skapats.

När varje satellit skapas läggs en referens till den till i arrayen `satellites`. När funktionen `doEveryFrame()` anropas uppdateras den till alla satelliter i arrayen.

Uppdatera satellitpositionen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Funktionen `doEveryFrame()` är hjärtat av programmets animeringsprocess. Den anropas för varje bildruta med samma frekvens som bildrutehastigheten i SWF-filen. Eftersom variablerna för ritåtgärden ändras en aning för varje bildruta ger det intrycket av animering.

Den första funktionen rensar allt som tidigare ritats och ritar om bakgrunden. Sedan skapas en slinga genom varje satellitbehållare och egenskapen `position` ökas för varje satellit. Egenskaperna `radius` och `orbitRadius` uppdateras om de har ändrats genom användarinteraktivitet från kontrollpanelen. Slutligen uppdateras satelliten till sin nya position genom ett anrop till metoden `draw()` i klassen Satellite.

Observera att räknaren, `i`, bara ökar värdet upp till variabeln `visibleSatellites`. Detta beror på, att om användaren har begränsat antalet satelliter som visas med kontrollpanelen, ska de resterande satelliterna i slingan inte ritas om utan döljas. Detta händer i en slinga som omedelbart följer slingan som ritningen görs i.

När funktionen doEveryFrame() är klar uppdateras antalet `visibleSatellites` i position på skärmen.

Svara på användarinteraktivitet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Användarinteraktivitet görs via kontrollpanelen och hanteras av klassen ControlPanel. Klassen ställer in en avlyssnare tillsammans med det lägsta värdet, högsta värdet och standardvärdet för varje reglage. När användaren flyttar reglagen anropas funktionen `changeSetting()`. Funktionen uppdaterar kontrollpanelens egenskaper. Om ändringen kräver att visningen ritas om skickas en händelse som sedan hanteras i huvudprogramfilen. När kontrollpanelens inställningar ändras ritar funktionen `doEveryFrame()` om varje satellit med de uppdaterade variablerna.

Ytterligare anpassning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det här exemplet är bara en grundläggande schematisk representation av hur man skapar visuellt innehåll med Drawing API. Relativt få kodrader används för att skapa en interaktiv upplevelse som ser ganska komplicerad ut. Trots det kan exemplet utökas med mindre ändringar. Några idéer:

- Funktionen `doEveryFrame()` skulle kunna inkrementera satellitens färgvärde.
- Funktionen `doEveryFrame()` skulle kunna krympa eller expandera satellitens radie över tiden.
- Satellitens radie behöver inte vara en cirkel. Klassen Math skulle kunna användas för att flytta den i till exempel en sinusvågrörelse.
- Satelliterna skulle kunna ha träffdetektering med andra satelliter.

Drawing API kan användas som ett alternativ till att skapa visuella effekter i Flash-utvecklingsmiljön för att rita grundläggande former vid körning. Det kan emellertid också användas för att skapa visuella effekter med variation och omfång som inte går att göra för hand. Med Drawing API och lite matematik kan en ActionScript-utvecklare ge liv åt många oväntade skapelser.

Använda avancerat Drawing API

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I Flash Player 10, Adobe AIR 1.5 och senare Flash-körningar finns stöd för en avancerad uppsättning ritfunktioner. Förbättringarna i rit-API:t för dessa körningar utökar ritmetoderna från tidigare versioner så att du kan skapa datauppsättningar för att generera former, ändra former under körning och skapa tredimensionella effekter. Förbättringarna i rit-API:t sammanför befintliga metoder i alternativa kommandon. De nya kommandona använder vektorarrayer och uppräkningsklasser för att tillhandahålla datauppsättningar för ritmetoder. Om du använder vektorarrayer kan mer komplexa former återges snabbare och utvecklare kan ändra arrayvärdena genom programmering, vilket gör det möjligt att återge dynamiska former under körning.

Ritfunktionerna som ingår från och med Flash Player 10 beskrivs i följande avsnitt: ”Rita banor” på sidan 236, ”Definiera vridningsregler” på sidan 237, ”Använda grafiska dataklasser” på sidan 239 och ”Om att använda drawTriangles()” på sidan 242.

Här nedan beskrivs uppgifter som du troligtvis vill utföra med hjälp av det avancerade Drawing API i ActionScript:

- Lagra data för ritmetoder med hjälp av Vector-objekt
- Definiera banor för att rita former genom programmering i en enda åtgärd
- Definiera vridregler för att bestämma hur överlappande former fylls
- Läsa ett visningsobjekts vektorgrafikinnehåll, till exempel för att serialisera och spara grafikdata, generera en Sprite-mall vid körning eller rita en kopia av vektorgrafikinnehållet
- Använda trianglar och ritmetoder för tredimensionella effekter

Viktiga termer och begrepp

Följande är en lista över viktiga begrepp som används i det här avsnittet:

- Vector: En array med värden av samma datatyp. Ett Vector-objekt kan lagra en array med värden som ritmetoder använder för att skapa linjer och former med ett enda kommando. Mer information om Vector-objekt finns i ”Indexerade arrayer” på sidan 25.
- Bana: En bana består av ett eller flera raka eller böjda segment. Början och slutet av varje segment är markerade med koordinater, som kan liknas vid nålar som håller fast en bit snöre. En bana kan vara stängd (t.ex. en cirkel) eller öppen med distinkta slutpunkter (t.ex. en vågig linje).
- Vridning: Riktningen för en bana som tolkas vid återgivningen. Den kan vara positiv (medurs) eller negativ (moturs).
- GraphicsStroke: En klass för inställning av linjeformat. Termen stroke (linje) är inte en del av förbättringarna i rit-API:t, men däremot användningen av en klass för att ange ett linjeformat med en egen fill-egenskap. Du kan justera ett linjeformat dynamiskt med klassen GraphicsStroke.

- Fill-objekt: Objekt som skapas med hjälp av visningsklasser som flash.display.GraphicsBitmapFill och flash.display.GraphicsGradientFill, som skickas till ritkommandot `Graphics.drawGraphicsData()`. Fill-objekt och de förbättrade ritkommandona resulterar i en mer objektorienterad programmeringsmetod för replikering av `Graphics.beginBitmapFill()` och `Graphics.beginGradientFill()`.

Rita banor

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I avsnittet om att rita linjer och kurvor (se "Rita linjer och kurvor" på sidan 223) introduceras kommandon för att rita en enskild linje (`Graphics.lineTo()`) eller kurva (`Graphics.curveTo()`) och sedan flytta linjen till en annan punkt (`Graphics.moveTo()`) för att skapa en form. Metoderna `Graphics.drawPath()` och `Graphics.drawTriangles()` accepterar en uppsättning med objekt som motsvarar samma ritkommandon som en parameter. Med de här metoderna kan du skapa en serie av `Graphics.lineTo()`-, `Graphics.curveTo()`- eller `Graphics.moveTo()`-kommandon som Flash-miljön kör i en enda sats.

Uppräkningsklassen GraphicsPathCommand definierar en uppsättning konstanter som motsvarar ritkommandon. Du skickar en serie med dessa konstanter (omslutna av en Vector-instans) som en parameter för metoden `Graphics.drawPath()`. Sedan kan du med ett enda kommando återge hela formen eller flera former. Du kan också ändra värdena som skickas till dessa metoder om du vill ändra en befintlig form.

Utöver vektorn med ritkommandon behöver metoden `drawPath()` även en uppsättning koordinater som motsvarar koordinaterna för varje enskilt ritkommando. Skapa en Vector-instans som innehåller koordinater (Number-instanser) och skicka den till metoden `drawPath()` som det andra (`data`)-argumentet.

Obs! *Värdena i vektorn är inte Point-objekt. Vektorn är en serie siffror där varje grupp med två siffror representerar ett par med x/y-koordinater.*

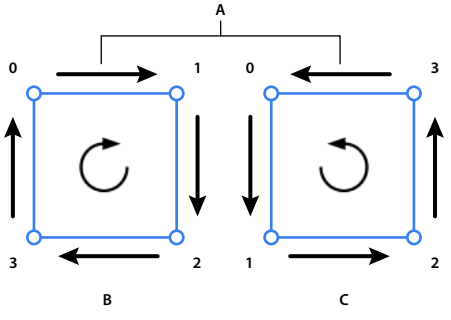
Metoden `Graphics.drawPath()` matchar varje kommando mot respektive punktvärde (en mängd med två eller flera siffror) för att skapa en bana i Graphics-objektet:

```
package
{
    import flash.display.*;

    public class DrawPathExample extends Sprite
    {
        public function DrawPathExample(){

            var squareCommands:Vector.<int> = new Vector.<int>(5, true);
            squareCommands[0] = GraphicsPathCommand.MOVE_TO;
            squareCommands[1] = GraphicsPathCommand.LINE_TO;
            squareCommands[2] = GraphicsPathCommand.LINE_TO;
            squareCommands[3] = GraphicsPathCommand.LINE_TO;
            squareCommands[4] = GraphicsPathCommand.LINE_TO;

            var squareCoord:Vector.<Number> = new Vector.<Number>(10, true);
            squareCoord[0] = 20; //x
            squareCoord[1] = 10; //y
            squareCoord[2] = 50;
            squareCoord[3] = 10;
            squareCoord[4] = 50;
            squareCoord[5] = 40;
            squareCoord[6] = 20;
            squareCoord[7] = 40;
            squareCoord[8] = 20;
            squareCoord[9] = 10;

            graphics.beginFill(0x442266);//set the color
            graphics.drawPath(squareCommands, squareCoord);
        }
    }
}
```

Definiera vridningsregler

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I de förbättrade funktionerna för rit-API ingår även ett nytt koncept för banvridning, det vill säga banans riktning. Vridningen för en bana är positiv (medurs) eller negativ (moturs). Vridningen bestäms utifrån ordningen som återgivningen tolkar de koordinater som anges av vektorn för dataparametern.



Positiv och negativ vridning
***A.** Pilar som anger ritriktning* ***B.** Positiv vridning (medsols)* ***C.** Negativ vridning (motsols)*

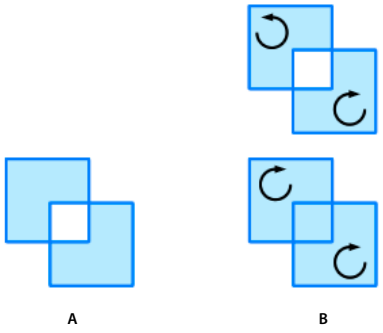
Lägg också märke till att metoden `Graphics.drawPath()` har en valfri tredje parameter med namnet ”winding”:

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

I det här sammanhanget är den tredje parametern en sträng eller en konstant som anger vridnings- eller fyllningsregeln för banor som överlappar varandra. (Konstanta värden definieras i klassen GraphicsPathWinding som `GraphicsPathWinding.EVEN_ODD` eller `GraphicsPathWinding.NON_ZERO`.) Vridningsregeln är viktig när banor överlappar.

Standardvridningsregeln är regeln jämn-udda och är den regel som används av det äldre rit-API:t. Jämn/udda-regeln är också standardregel för metoden `Graphics.drawPath()`. Med jämn/udda-regeln växlar alla banor som överlappar mellan öppna och stängda fyllningar. Om två fyrkanter som är ritade med samma fyllning överlappar varandra fylls överlappningsområdet. Om två områden gränsar till varandra är båda inte fyllda respektive ofyllda samtidigt.

Regeln inte-noll, å andra sidan, är beroende av vridningen (ritriktningen) för att bestämma om områden som definieras av överlappande banor ska fyllas i eller inte. När banor med olika vridning överlappar, definieras området som ofyllt, på samma sätt som vid udda/jämn-regeln. För banor med samma vridning fylls området som borde vara ofyllt:

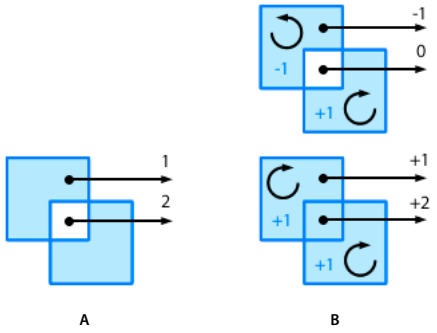


Vridningsregler för överlappande områden
A. *Vridningsregeln jämn/udda* ***B.*** *Vridningsregeln inte-noll*

Namn på vridningsregler

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Namnen beskriver en mer specifik regel som definierar hur dessa fyllningar hanteras. Medurs vridna banor tilldelas värdet +1, medan moturs vridna banor tilldelas värdet -1. Börja vid en punkt inom ett omslutet område i en form, och rita en linje från den punkten som sträcker ut oändligt. Antalet gånger som den linjen korsar en bana, och de kombinerade värdena för dessa banor, används för att fastställa fyllningen. När det gäller udda/jämn-vridning används antalet gånger som linjen korsar en bana. När antalet är udda, fylls området i. När antalet är jämnt, fylls området inte i. När det gäller inte-noll-vridning används värdena som tilldelas banorna. När de kombinerade värdena för banan inte är 0, fylls området i. När det kombinerade värdet är 0, fylls området inte i.



Antal och fyllningar vid vridningsregler
A. *Vridningsregeln jämn/udda* ***B.*** *Vridningsregeln inte-noll*

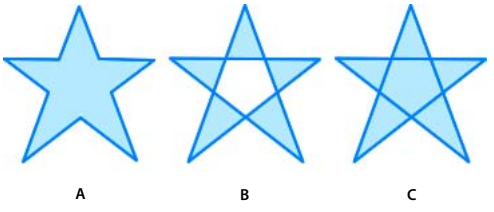
Använda vridningsregler

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Dessa fyllningsregler är komplicerade, men i vissa fall nödvändiga. Tänk dig att du ritar en stjärnform. Med den vanliga udda/jämn-regeln krävs tio olika linjer för formen. Med regeln icke-noll minskas antalet linjer till fem. Här är ett ActionScript för en stjärna med fem linjer och vridningsregeln inte-noll:

```
graphics.beginFill(0x60A0FF);
graphics.drawPath( Vector.<int>([1,2,2,2,2]),  Vector.<Number>([66,10, 23,127, 122,50, 10,49,
109,127]),   GraphicsPathWinding.NON_ZERO);
```

Och här är stjärnformen:



En stjärnform med olika vridningsregler
A. *Udda/jämn, 10 linjer* ***B.*** *Udda/jämn, 5 linjer* ***C.*** *Inte-noll, 5 linjer*

Eftersom bilder animeras eller används som textur i tredimensionella objekt och överlappar, blir vridningsreglerna ännu viktigare.

Använda grafiska dataklasser

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Det förbättrade rit-API:t inkluderar en uppsättning klasser i flash.display-paketet som implementerar gränssnittet IGraphicsData. De här klasserna fungerar som värdeobjekt (databehållare) som representerar rit-API:ts ritmetoder.

Följande klasser implementerar IGraphicsData-gränssnittet:

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath

- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Med de här klasserna kan du lagra en fullständig ritning i ett Vector-objekt av IGraphicsData-typ (Vector.<IGraphicsData>). Du kan sedan återanvända dessa grafikdata som datakälla för andra forminstanser eller för att lagra ritinformation för framtida bruk.

Observera att det finns flera fyllningsklasser för varje fyllningsformat, men bara en linjeklass. ActionScript innehåller bara en IGraphicsData-linjeklass eftersom linjeklasser använder fyllningsklasserna när formatet definieras. Varje linje definieras i själva verket av en kombination av klassen Stroke och en fyllningsklass. API:et för dessa grafiska dataklasser avspeglar annars metoderna som de representerar i klassen flash.display.Graphics:

Graphics-metod	**Motsvarande klass**
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

Dessutom har klassen GraphicsPath egna `GraphicsPath.moveTo()`-, `GraphicsPath.lineTo()`-, `GraphicsPath.curveTo()`-, `GraphicsPath.wideLineTo()`- och `GraphicsPath.wideMoveTo()`-verktygsmetoder så att dessa kommandon enkelt kan definieras för en GraphicsPath-instans. Dessa verktygsmetoder gör det enklare att definiera och uppdatera kommandona och datavärdena direkt.

Rita med vektorgrafikdata

När du har en samling IGraphicsData-instanser använder du Graphics-klassens `drawGraphicsData()`-metod för att återge grafiken. Metoden `drawGraphicsData()` utför en rad ritinstruktioner från en vektor med IGraphicsData-instanser i ordningsföljd:

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70, 70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(GraphicsPathCommand.MOVE_TO, GraphicsPathCommand.LINE_TO,
GraphicsPathCommand.LINE_TO);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

Genom att ändra ett värde i banan som används av teckningen i exemplet, kan du rita om formen flera gånger och få en mer komplex bild:

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

IGraphicsData-objekt kan definiera fyllnings- och linjeformat, men fyllnings- och linjeformat är inget krav. Graphics-klassmetoder kan med andra ord användas för att ställa in format medan IGraphicsData-objekt kan användas för att rita en sparad mängd med banor, och tvärtom.

***Obs!** Använd metoden `Graphics.clear()` om du vill ta bort en tidigare teckning innan du påbörjar en ny, såvida du inte lägger till något i den ursprungliga teckningen som i exemplet ovan. När du ändrar en enskild del av en bana eller mängd med IGraphicsData-objekt, måste du rita om hela teckningen för att kunna se ändringarna.*

När du använder grafikdataklasser återges fyllningen när tre eller fler punkter ritas, eftersom formen är stängd vid den punkten. Även om fyllningen stängs gör inte linjen det, och dess beteende är ett annat än när du använder flera `Graphics.lineTo()`- eller `Graphics.moveTo()`-kommandon.

Läsa vektorgrafikdata

Flash Player 11.6 och senare, Adobe AIR 3.6 och senare

I Flash Player 11.6 samt Adobe AIR 3.6 och senare versioner kan du, utöver att rita vektorinnehåll till ett visningsobjekt, använda Graphics-klassens `readGraphicsData()`-metod för att hämta en datarepresentation av ett visningsobjekts vektorgrafikinnehåll. Detta är praktiskt när du vill skapa en ögonblicksbild av en bild som du till exempel vill spara, kopiera eller skapa en Sprite-mall för.

När `readGraphicsData()`-metoden anropas returneras en Vector-instans med IGraphicsData-objekt. Detta är samma objekt som används för att rita vektorgrafik med metoden `drawGraphicsData()`.

Det finns ett antal begräsningar när det gäller att läsa vektorgrafik med metoden `readGraphicsData()`. Mer information om detta finns i avsnittet om readGraphicsData() i referenshandboken för ActionScript.

Om att använda drawTriangles()

Flash Player 10 och senare, Adobe AIR 1.5 och senare

En annan ny avancerad metod i Flash Player 10 och Adobe AIR 1.5, `Graphics.drawTriangles()`, liknar metoden `Graphics.drawPath()`. Metoden `Graphics.drawTriangles()` använder också en Vector.<Number>-objekt för att ange punktplatser för ritning av en bana.

Det verkliga syftet med metoden `Graphics.drawTriangles()` är emellertid att möjliggöra tredimensionella effekter i ActionScript. Information om hur du använder `Graphics.drawTriangles()` för att skapa tredimensionella effekter finns i ”Använda trianglar för 3D-effekter” på sidan 363.

Kapitel 13: Arbeta med bitmappar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med ActionScript 3.0 kan du, förutom att använda funktioner för vektorritning, skapa bitmappsbilder eller ändra pixeldata i externa bitmappsbilder som lästs in till en SWF-fil. Eftersom du kan komma åt och ändra individuella pixelvärden kan du skapa egna filterliknande bildeffekter och använda de inbyggda brusfunktionerna för att skapa texturer och slumpvist brus.

- Renaun Erickson: Rendering game assets in ActionScript using blitting techniques
- Bitmap programming: Kapitel 26 i Essential ActionScript 3 av Colin Moock (O'Reilly Media, 2007)
- Mike Jones: Working with Sprites in Pushbutton Engine
- Flash & Math: Pixel Particles Made Simple
- Flixel

Grunderna för arbete med bitmappar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du arbetar med digitala bilder kommer du att upptäcka två grafikhuvudtyper: bitmapp och vektor. Bitmappsgrafik, som också kallas rastergrafik, är uppbyggd av små fyrkanter (pixlar) som ordnats i ett rektangulärt rutnätsformat. Vektorgrafik är uppbyggd av matematiskt genererade geometriska former som linjer, kurvor och polygoner.

Bitmappsbilder definieras av bildens bredd och höjd, mätt i pixlar, och antalet bitar som finns i respektive pixel, vilket anger antalet färger en pixel kan innehålla. Om en bitmappsbild använder RGB-färgmodellen består pixeln av tre byte: röd, grön och blå. Varje byte innehåller ett värde mellan 0 och 255. Om olika byte kombineras i pixeln skapas en färg på samma sätt som när en artist blandar målarfärger. Med en pixel som innehåller bytevärdena röd-255, grön-102 och blå-0 skapas en livfull orange färg.

Kvaliteten på en bitmappsbild bestäms av bildens upplösning i kombination med färgdjupets bitvärde. *Upplösning relateras till antalet pixlar i en bild.* Ju högre antal pixlar desto högre upplösning och finare bild. *Färgdjup relateras till den mängd information en pixel kan innehålla.* En bild som har ett färgdjup på 16 bitar per pixel kan inte representera samma antal färger som en bild som har ett färgdjup på 48 bitar. 48-bitarsbilden får alltså jämnare skuggor än 16-bitarsbilden.

Eftersom bitmappsgrafik är upplösningsberoende skalförändras den inte så bra. Detta märks mest när bitmappsbilder skalförändras uppåt. En bitmappsbild som skalförändras uppåt förlorar detaljer och kvalitet.

Bitmappsbilders format

Bitmappsbilder grupperas i ett antal vanliga filformat. I dessa format används olika typer av komprimeringsalgoritmer för att minska filstorleken, men också för att optimera bildkvaliteten beroende på vad bilden ska användas till. De bitmappformat som stöds av Adobes körningsmiljöer är BMP, GIF, JPG, PNG och TIFF.

BMP

BMP-formatet (bitmapp) är standardbildformatet som används i Microsofts operativsystem Windows. Det använder inte någon form av komprimeringsalgoritm och resulterar därför oftast i mycket stora filer.

GIF

GIF (Graphics Interchange Format) utvecklades av CompuServe år 1987 för att överföra bilder med 256 färger (8-bitars färg). Formatet ger liten filstorlek och är idealiskt för webbaserade bilder. På grund av formatets begränsade färgpalett passar GIF-bilder vanligtvis inte till fotografier, eftersom dessa kräver jämna skuggor och övertoningsfärger. GIF-bilder tillåter enkelbitsgenomskinlighet, vilket gör att färgerna kan mappas som klara (eller genomskinliga). Detta gör att en webbsidas bakgrundsfärg visas genom bilden där genomskinligheten har mappats.

JPEG

I JPEG-formatet (skrivs ofta JPG), som utvecklats av Joint Photographic Experts Group (JPEG), används en förstörande komprimeringsalgoritm som tillåter 24-bitars färgdjup med liten filstorlek. Med förstörande komprimering menas att varje gång bilden sparas förlorar bilden i kvalitet och i data men filen blir mindre. JPEG-formatet är idealiskt för fotografier eftersom miljoner färger kan visas. Eftersom du kan styra bildens komprimeringsgrad kan du manipulera bildkvaliteten och filstorleken.

PNG

PNG-formatet (Portable Network Graphics) skapas som ett öppenkälla-alternativ till det patenterade GIF-filformatet. PNG-filer kan använda upp till 64-bitars färgdjup, och upp till 16 miljoner färger. Eftersom PNG är ett ganska nytt format stöder äldre webbläsare inte PNG-filer. I motsats till JPG-filer använder PNG-filer icke-förstörande komprimering, vilket betyder att inga bilddata förloras när bilden sparas. PNG-filer stöder alltså alfagenomskinlighet, som tillåter upp till 256 genomskinlighetsnivåer.

TIFF

TIFF (Tagged Image File Format) var det vanligaste plattformsoberoende formatet innan PNG lanserades. Nackdelen med TIFF-formatet är att det finns många varianter, och därför ingen enskild läsare som kan hantera alla versioner. Dessutom har inga webbläsare för närvarande stöd för formatet. TIFF kan använda förstörande eller icke-förstörande komprimering, och kan hantera enhetsspecifika färgrymder (till exempel CMYK).

Genomskinliga och ogenomskinliga bitmappar

I bitmappsbilder med formatet GIF eller PNG kan en extra byte (alfakanal) läggas till i varje pixel. Den här extra pixelbyten representerar pixelns genomskinlighetsvärde.

I GIF-bilder kan du använda enkelbitsgenomskinlighet, vilket betyder att du kan ange att en enstaka färg på en 256-färgspalett ska vara genomskinlig. PNG-bilder kan å andra sidan ha upp till 256 genomskinlighetsnivåer. Den här funktionen är särskilt bra när bilder eller text ska blandas in i bakgrunden.

I ActionScript 3.0 replikeras den här extra genomskinlighetspixeln i klassen BitmapData. Precis som PNG-genomskinlighetsmodellen har ActionScript upp till 256 genomskinlighetsnivåer.

Viktiga termer och begrepp

Följande lista innehåller viktiga termer som du kommer att stöta på när du läser om bitmappar:

Alfa Genomskinlighetsnivån (eller rättare sagt opaciteten) i en färg eller en bild. Mängden alfa beskrivs ibland som *alfakanalsvärde*.

ARGB-färg Ett färgschema där varje pixels färg är en blandning av rött, grönt och blått, och genomskinligheten anges som ett alfavärde.

Färgkanal Vanligtvis representeras färger som en blandning av några enstaka grundfärger, vanligen (för datorgrafik) röd, grön och blå. Varje grundfärg betraktas som en färgkanal. Mängden färg i varje färgkanal som har blandats avgör den slutliga färgen.

Färgdjup Kallas också *bitdjup* och är mängden datorminne som reserveras för varje pixel, vilket i sin tur avgör antalet möjliga färger som kan visas i en bild.

Pixel Den minsta informationsenheten i en bitmappsbild, egentligen en färgpunkt.

Upplösning En bilds pixelmått, som bestämmer nivån på finkorniga detaljer i bilden. Upplösningen anges ofta som antalet pixlar för bildens bredd och höjd.

RGB-färg Ett färgschema där varje pixels färg representeras som en blandning av rött, grönt och blått.

Klasserna Bitmap och BitmapData

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du arbetar med bitmappbilder är de viktigaste ActionScript 3.0-klasserna Bitmap, som används för att visa bitmappbilder på skärmen, och BitmapData, som används för att öppna och hantera bitmappbildens rådata.

Fler hjälpavsnitt

flash.display.Bitmap

flash.display.BitmapData

Klassen Bitmap

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Bitmapp, som är en underklass till klassen DisplayObject, är huvudklassen i ActionScript 3.0 för visning av bitmappsbilder. De här bilderna har kanske lästs in via klassen flash.display.Loader eller skapats dynamiskt med konstruktorn `Bitmap()`. När en bild läses in från en extern källa kan ett Bitmap-objekt bara använda GIF-, JPEG- eller PNG-bilder. När Bitmap-instansen instansierats betraktas den som en wrapper för ett BitmapData-objekt som måste återges på scenen. Eftersom en Bitmap-instans är ett visningsobjekt, kan alla egenskaper och funktioner för visningsobjekt också användas för att ändra en Bitmap-instans. Mer information om hur du arbetar med visningsobjekt finns i ”Visningsprogrammering” på sidan 151.

Pixelfästning och utjämning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Bitmapp, har förutom de funktioner som är vanliga i alla visningsobjekt, vissa andra funktioner som är speciella för bitmappsbilder.

Egenskapen `pixelSnapping` för klassen Bitmap avgör om ett Bitmap-objekt fästs mot dess närmsta pixel eller inte. Den här egenskapen godkänner en av tre konstanter som definierats i klassen PixelSnapping: `ALWAYS`, `AUTO` och `NEVER`.

Syntax för pixelfästning är:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Bitmappsbilder som skalförändras blir ofta oskarpa och förvrängda. Du kan använda egenskapen `smoothing` för klassen BitmapData för att minska förvrängningen. Om den här booleska egenskapen har värdet `true` utjämnas eller kantutjämnas pixlarna i bilden när den skalförändras. Detta ger bilden ett klarare och mera naturligt utseende.

Klassen BitmapData

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen BitmapData, som finns i paketet flash.display, kan liknas vid en ögonblicksbild av de pixlar som ingår i en inläst eller dynamiskt skapad bitmappsbild. Ögonblicksbilden representeras av en array av pixeldata i objektet. Klassen BitmapData innehåller också en serie inbyggda metoder som kan vara bra när du skapar och ändrar pixeldata.

Använd följande kod när du instansierar ett BitmapData-objekt:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Parametrarna `width` och `height` anger bitmappens storlek. Från och med AIR 3 och Flash Player 11 har storleksgränserna för BitmapData-objektet tagits bort. Den största tillåtna storleken på en bitmapp beror på operativsystemet.

I AIR 1.5 och Flash Player 10 är maxstorleken för ett BitmapData-objekt 8 191 pixlar i bredd eller höjd, och det totala antalet pixlar får inte överskrida 16 777 215 pixlar. (Om ett BitmapData-objekt är 8 191 pixlar brett kan det därför bara vara 2 048 pixlar högt.) I Flash Player 9 och tidigare och i AIR 1.1 och tidigare är begränsningen 2 880 pixlar högt och 2 880 pixlar brett.

Parametern `transparent` anger om bitmappsdata ska innehålla en alfakanal (`true`) eller inte (`false`). Parametern `fillColor` är ett 32-bitarsvärde som anger bakgrundsfärgen och genomskinlighetsvärdet (om det är `true`). I följande exempel skapas ett BitmapData-objekt med orange bakgrund som är 50 procent genomskinligt:

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Om du vill återge ett nyligen skapat BitmapData-objekt på skärmen, kopplar du det eller omsluter du det i en Bitmap-instans. Detta gör du genom att skicka BitmapData-objektet som en parameter för objektets konstruktor, eller genom att koppla det till egenskapen `BitmapData` för en befintlig Bitmap-instans. Du måste också lägga till Bitmap-instansen i visningslistan genom att anropa metoderna `addChild()` eller `addChildAt()` för visningsobjektets behållare som ska innehålla Bitmap-instansen. Mer information om hur du arbetar med visningslistan finns i ”Lägga till visningsobjekt till visningslistan” på sidan 159.

I följande exempel skapas ett BitmapData-objekt med röd fyllning som visas i en Bitmap-instans:

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Ändra pixlar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen BitmapData innehåller en serie metoder som du använder för att ändra pixeldatavärden.

Ändra individuella pixlar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du ändrar utseendet på en bitmappsbild på pixelnivå, måste du först hämta färgvärdena för pixlarna inom det område som du vill ändra. Använd metoden `getPixel()` för att läsa dessa pixelvärden.

Med metoden `getPixel()` hämtas ett RGB-värde från en serie x-, y-koordinater (pixel) som skickats som parametrar. Om någon av de pixlar du vill ändra innehåller genomskinlighetsinformation (alfakanal) måste du använda metoden `getPixel32()`. Metoden hämtar också ett RGB-värde, men det värde som returneras med `getPixel()` innehåller, i motsats till det värde som hämtas med `getPixel32()` ytterligare data som representerar alfakanalsvärdet (genomskinlighet) för de valda pixlarna.

Om du bara vill ändra färg eller genomskinlighet på en pixel som finns i en bitmapp, kan du använda metoden `setPixel()` eller `setPixel32()`. Om du vill ange en pixels färg skickar du x-, y-koordinaterna och färgvärdet till någon av dessa metoder.

I följande exempel används `setPixel()` för att rita ett kors på en grön BitmapData-bakgrund. Därefter används `getPixel()` för att hämta färgvärdet från pixeln på koordinaten 50, 50, och det returnerade värdet kalkeras.

```
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));
```

Om du vill läsa värdet för en grupp pixlar, i motsats till en enstaka pixel, använder du metoden `getPixels()`. Med den här metoden genereras en byte-array från ett rektangulärt område med pixeldata som skickats som en parameter. Varje element i byte-arrayen (alltså pixelvärdena) är heltal utan tecken, 32-bitars icke-multiplicerade pixelvärden.

Om du omvänt vill ändra (eller ange) värdet för en grupp pixlar använder du metoden `setPixels()`. Den här metoden förväntar sig två parametrar (`rect` och `inputByteArray`) som kombineras för att mata ut ett rektangulärt område (`rect`) av pixeldata (`inputByteArray`).

När data läses (och skrivs) från `inputByteArray` anropas metoden `ByteArray.readUnsignedInt()` för varje pixel i arrayen. Om `inputByteArray` av någon anledning inte innehåller ett fullständigt rektangelvärde med pixeldata, stoppas hanteringen av bilddata av metoden.

Kom ihåg att byte-arrayen förväntar sig 32-bitars alfa, röd, grön, blå (ARGB) både vid hämtning och inställning av pixeldata.

I följande exempel används metoderna `getPixels()` och `setPixels()` för att kopiera pixlar mellan olika BitmapData-objekt:

```
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;
```

Kollisionsidentifiering på pixelnivå

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `BitmapData.hitTest()` identifieras kollisioner på pixelnivå mellan bitmappsdata och andra objekt eller punkter.

Metoden `BitmapData.hitTest()` godkänner fem parametrar:

- `firstPoint` (Point): Den här parametern refererar till pixelpositionen för det övre vänstra hörnet av den första BitmapData på vilket träfftestet utförs.
- `firstAlphaThreshold` (uint): Den här parametern anger det högsta alfakanalsvärdet som räknas som ogenomskinligt för det här träfftestet.
- `secondObject` (Object): Den här parametern representerar effektområdet. Objektet `secondObject` kan vara ett Rectangle-, Point-, Bitmap- eller BitmapData-objekt. Det här objektet representerar träffområdet som kollisionsidentifieringen utförs på.
- `secondBitmapDataPoint` (Point): Den här valfria parametern används för att definiera en pixelplats i det andra BitmapData-objektet. Parametern används bara om värdet för `secondObject` är ett BitmapData-objekt. Standardvärdet är `null`.
- `secondAlphaThreshold` (uint): Den här valfria parametern representerar det högsta alfakanalsvärdet som räknas som ogenomskinligt i det andra BitmapData-objektet. Standardvärdet är 1. Den här parametern används bara när värdet för `secondObject` är ett BitmapData-objekt och båda BitmapData-objekten är genomskinliga.

När du utför en kollisionsidentifiering på ogenomskinliga bilder bör du tänka på att ActionScript hanterar bilden som om den är en fullständigt ogenomskinlig rektangel (eller begränsningsram). När du utför ett träfftest på genomskinliga bilder måste båda bilderna vara genomskinliga. ActionScript använder dessutom alfatröskelparametrarna för att bestämma vid vilken punkt pixlarna ska ändras från genomskinliga till ogenomskinliga.

I följande exempel skapas tre bitmappsbilder och eftersöks pixelkollisioner med hjälp av två olika kollisionspunkter (en returnerar false och en annan true):

```
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
```

Kopiera bitmappsdata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan välja mellan olika metoder för att kopiera bitmappsdata från en bild till en annan: `clone()`, `copyPixels()`, `copyChannel()`, `draw()` eller `drawWithQuality()` (metoden `drawWithQuality`finns i Flash Player 11.3 och senare; AIR 3.3 och senare).

Som namnet anger kan du med metoden `clone()` klona, eller sampla, bitmappsdata mellan olika BitmapData-objekt. När metoden anropas returneras ett nytt BitmapData-objekt som är en exakt klon av den ursprungliga instans som det kopierats från.

I följande exempel klonas en kopia av en orange (överordnad) fyrkant och klonen placeras bredvid den ursprungliga överordnade fyrkanten:

```
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;
```

Med metoden `copyPixels()` kan du snabbt och lätt kopiera pixlar mellan olika BitmapData-objekt. Metoden tar en rektangulär ögonblicksbild (definieras av parametern `sourceRect`) av källbilden och kopierar den till ett annat rektangulärt område (av samma storlek). Platsen för den nyligen ”inklistrade” rektangeln definieras i parametern `destPoint`.

Metoden `copyChannel()` samplar ett fördefinierat färgkanalsvärde (alfa, röd, grön eller blå) från ett BitmapData-källobjekt och kopierar det till en kanal för ett BitmapData-destinationsobjekt. Anrop till den här metoden påverkar inte de andra kanalerna i destinationsobjektet.

Metoderna `draw()` och `drawWithQuality()` ritar (eller återger) det grafiska innehållet från en käll-sprite, ett filmklipp eller något annat visningsobjekt till en ny bitmapp. Om du använder parametrarna `matrix`, `colorTransform`, `blendMode` och destinationsparametern `clipRect` kan du ändra det sätt som den nya bitmappen återges på. Med den här metoden används vektorrenderaren i Flash Player och AIR för att generera data.

När du anropar `draw()` eller `drawWithQuality()` skickar du källobjektet (en sprite, ett filmklipp eller något annat visningsobjekt) som den första parametern, enligt nedanstående:

```
myBitmap.draw(movieClip);
```

Om några omformningar (färg, matris o.s.v.) har tillämpats på källobjektet efter att det ursprungligen lästs in, kopieras dessa inte över till det nya objektet. Om du vill kopiera omformningarna till den nya bitmappen måste du kopiera värdet för egenskapen `transform` från ursprungsobjektet till egenskapen `transform` för det bitmappsobjekt som använder det nya bitmappsobjektet.

Komprimera bitmappdata

Flash Player 11.3 och senare, AIR 3.3 och senare

Med metoden `flash.display.BitmapData.encode()` kan du komprimera bitmappdata till ett av följande bildformat:

- **PNG** – Använder PNG-komprimering, med snabb komprimering om så önskas, vilket prioriterar komprimeringshastighet i stället för filstorlek. Om du vill använda PNG-komprimering skickar du ett nytt `flash.display.PNGEncoderOptions`-objekt som den andra parametern i metoden `BitmapData.encode()`.
- **JPEG** – Använder JPEG-komprimering, där bildkvalitet kan anges om så önskas. Om du vill använda JPEG-komprimering skickar du ett nytt `flash.display.JPEGEncoderOptions`-objekt som den andra parametern i metoden `BitmapData.encode()`.

- **JPEGXR** – Använder JPEGXR-komprimering (JPEG Extended Range), där inställningar för färgkanal, förstörelsegrad och entropi kan anges om så önskas. Om du vill använda JPEGXR-komprimering skickar du ett nytt `flash.display.JPEGXREncoderOptions`-objekt som den andra parametern i metoden `BitmapData.encode()`.

Du kan använda den här funktionen för bildbearbetning som en del i ett arbetsflöde för överföring till eller från en server.

I följande exempelfragment komprimeras ett BitmapData-objekt med `JPEGEncoderOptions`:

```
// Compress a BitmapData object as a JPEG file.
var bitmapData:BitmapData = new BitmapData(640,480,false,0x00FF00);
var byteArray:ByteArray = new ByteArray();
bitmapData.encode(new Rectangle(0,0,640,480), new flash.display.JPEGEncoderOptions(),
byteArray);
```

Skapa texturer med brusfunktioner

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill ändra utseendet på en bitmapp kan du använda en bruseffekt med hjälp av metoden `noise()` eller `perlinNoise()`. En bruseffekt kan liknas vid den statiskhet som uppträder på en ej avstämd teveskärm.

Om du vill tillämpa en bruseffekt på en bitmapp använder du metoden `noise()`. Metoden använder ett godtyckligt färgvärde för pixlarna inom ett angivet område av en bitmappsbild.

Metoden godkänner fem parametrar:

- `randomSeed` (int): ett godtyckligt startvärde som anger mönstret. Trots namnet skapas samma resultat om samma värde skickas. Om du vill få ett sant godtyckligt resultat använder du metoden `Math.random()` för att skicka ett godtyckligt värde för den här parametern.
- `low` (uint): den här parametern refererar till det lägsta värde som ska genereras för varje pixel (0 till 255). Standardvärdet är 0. Om du anger ett lägre värde får du ett mörkare brusmönster och om du anger ett högre värde får du ett ljusare mönster.
- `high` (uint): den här parametern refererar till det högsta värde som ska genereras för varje pixel (0 till 255). Standardvärdet är 255. Om du anger ett lägre värde får du ett mörkare brusmönster och om du anger ett högre värde får du ett ljusare mönster.
- `channelOptions` (uint): den här parametern anger på vilken färgkanal för bitmappsobjektet som brusmönstret ska tillämpas. Värdet kan vara en kombination av några av de fyra ARGB-färgkanalsvärdena. Standardvärdet är 7.
- `grayScale` (Boolean): om värdet är `true` tillämpas värdet `randomSeed` på bitmappspixlarna, vilket effektivt tar bort alla färger från bilden. Alfakanalen påverkas inte av den här parametern. Standardvärdet är `false`.

I följande exempel skapas en bitmappsbild och tillämpas ett blått brusmönster:

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise1 extends Sprite
    {
        public function BitmapNoise1()
        {
            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);
        }
    }
}
```

Om du vill skapa en mer organisk textur använder du metoden `perlinNoise()`. Med metoden `perlinNoise()` skapas realistiska organiska texturer som är idealiska för rök, moln, vatten, eld och även explosioner.

Eftersom metoden `perlinNoise()` genereras av en algoritm använder den mindre minne än bitmappsbaserade texturer. Den kan däremot fortfarande påverka processoranvändningen, så att det tar längre tid att visa innehåll och skärmen ritas om långsammare än bildrutehastigheten, särskilt på gamla datorer. Detta beror huvudsakligen på flyttalsberäkningarna som måste utföras för att Perlin-brusalgoritmerna ska bearbetas.

Metoden godkänner nio parametrar (de första sex är obligatoriska):

- `baseX` (Number): anger x-värdet (storlek) för skapade mönster.
- `baseY` (Number): anger y-värdet (storlek) för skapade mönster.
- `numOctaves` (uint): antal oktaver eller enskilda brusfunktioner som ska kombineras för att detta brus ska uppstå. Ju fler oktaver desto högre detaljrikedom får bilderna, men detta kräver också längre bearbetningstid.
- `randomSeed` (int): det godtyckliga startnumret fungerar på samma sätt som i funktionen `noise()`. Om du vill få ett sant godtyckligt resultat använder du metoden `Math.random()` för att skicka ett godtyckligt värde för den här parametern.
- `stitch` (Boolean): Om värdet är `true` försöker metoden maska (eller jämna ut) övergångskanterna i bilden för att skapa sömlösa texturer när en bitmappsfyllning används.
- `fractalNoise` (Boolean): Den här parametern relateras till kanterna på de övertoningar som genereras av metoden. Om värdet är `true` genereras fraktalbrus som jämnar ut kanterna. Om värdet är `false` genereras turbulens. En bild med turbulens har synliga avbrott i övertoningen som kan göra den bättre på att visa skarpare visuella effekter, till exempel flammor eller vågor på havet.
- `channelOptions` (uint): Parametern `channelOptions` fungerar på samma sätt som i metoden `noise()`. Den anger vilken färgkanal (för bitmappen) som brusmönstret ska tillämpas på. Värdet kan vara en kombination av några av de fyra ARGB-färgkanalsvärdena. Standardvärdet är 7.
- `grayScale` (Boolean): Parametern `grayScale` fungerar på samma sätt som i metoden `noise()`. Om värdet är `true` tillämpas värdet `randomSeed` på bitmappspixlarna, vilket effektivt tar bort alla färger från bilden. Standardvärdet är `false`.
- `offsets` (Array): En array med punkter som motsvarar x- och y-förskjutningar för varje oktav. Om du ändrar förskjutningsvärdena kan du rulla lagren i bilden på ett jämnt sätt. Varje punkt i förskjutningsarrayen påverkar en viss oktavbrusfunktion. Standardvärdet är `null`.

I följande exempel skapas ett BitmapData-objekt på 150 x 150 pixlar som anropar metoden `perlinNoise()` för att generera en grön och blå molneffekt:

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise2 extends Sprite
    {
        public function BitmapNoise2()
        {
            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);
        }
    }
}
```

Rulla bitmappar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Anta att du har skapat program över gatukartor där du måste uppdatera vyn varje gång kartan flyttas (även om kartan bara flyttats några få pixlar).

Ett sätt att skapa den här funktionen är att återge en ny bild på nytt som innehåller den uppdaterade kartvyn varje gång användaren flyttar kartan. Du kan också skapa en stor enkelbild och använda metoden `scroll()`.

Metoden `scroll()` kopierar en skärmbitmapp och klistrar in den på en ny förskjutningsplats, som anges av parametrarna (`x`, `y`). Om en del av bitmappen finns utanför scenen, får det till resultat att bilden skiftas. Om detta kombineras med en timerfunktion (eller en `enterFrame`-händelse) kan du låta bilden visas som animering eller rullning.

I följande exempel används föregående Perlin-brusexempel och genereras en större bitmappsbild (tre fjärdedelar återges utanför scenen). Metoden `scroll()` tillämpas därefter, tillsammans med händelseavlyssnaren `enterFrame` som förskjuter bilden med en pixel diagonalt nedåt. Metoden anropas varje gång bildrutan anges och som resultat återges delarna som ligger utanför skärmen på scenen vartefter bilden rullas nedåt.

```
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);
}
```

Utnyttja mipmapping

Flash Player 9 och senare, Adobe AIR 1.0 och senare

MIP-mappar (som också kallas för *mipmappar*) är bitmappar som grupperas tillsammans och kopplas till en textur för att öka återgivningskvaliteten och prestandan under körning. Varje bitmappsbild i MIP-mappen är en version av den huvudsakliga bitmappsbilden, men innehåller inte lika många detaljer.

Du kan till exempel ha en mipmapp som omfattar en huvudbild på 64 x 64 pixlar vid högsta kvalitet. Bilder med lägre kvalitet i mipmappen är då 32 × 32, 16 × 16, 8 × 8, 4 × 4, 2 × 2 och 1 × 1 pixlar.

Texturuppspelning är möjligheten att läsa in den bitmapp som har lägst kvalitet först och sedan stegvis visa bitmappar med högre kvalitet efterhand som bitmapparna läses in. Eftersom bitmappar med lägre kvalitet är små, läses de in snabbare än huvudbilden. Därför kan programanvändarna visa bilden i ett program innan den huvudsakliga bilden med hög kvalitet har lästs in.

Flash Player 9.115.0. och senare versioner och AIR implementerar den här tekniken (processen kallas för *mipmapping*) genom att skapa optimerade versioner i olika skalor av respektive bitmapp (med början vid 50 %).

Flash Player 11.3 och AIR 3.3 har stöd för texturuppspelning via parametern `streamingLevels` i metoderna `Context3D.createCubeTexture()` och `Context3D.createTexture()`.

Med texturkomprimering kan du lagra komprimerade texturbilder direkt på GPU:n, vilket kräver mindre GPU-minne och minnesbandbredd. Komprimerade texturer komprimeras normalt offline och överförs till GPU:n i komprimerat format. Flash Player 11.4 och AIR 3.4 har dock stöd för texturkomprimering under körning vilket är användbart i vissa situationer, t.ex. vid återgivning av dynamiska texturer från vektorgrafik. Gör så här för att använda texturkomprimering under körning:

- Skapa texturobjektet genom att anropa metoden `Context3D.createTexture()`, och skicka antingen `flash.display3D.Context3DTextureFormat.COMPRESSED` eller `flash.display3D.Context3DTextureFormat.COMPRESSED_ALPHA` i den tredje parametern.

- Använd den `flash.display3D.textures.Texture`-instans som returneras av `createTexture()` och anropa antingen `flash.display3D.textures.Texture.uploadFromBitmapData()` eller `flash.display3D.textures.Texture.uploadFromByteArray()`. Dessa metoder överför och komprimerar texturen på samma gång.

MIP-mappar skapas för följande bitmapptyper:

- en bitmapp (JPEG-, GIF- eller PNG-filer) som visas med klassen Loader i ActionScript 3.0
- en bitmapp i biblioteket för ett Flash Professional-dokument
- ett BitmapData-objekt
- en bitmapp som visas med funktionen `loadMovie()` i ActionScript 2.0

MIP-mappar används inte på filtrerade objekt eller bitmappscachelagrade filmklipp. MIP-mappar används emellertid om du har bitmappsomformningar i ett filtrerat visningsobjekt, även om bitmappen finns i maskat innehåll.

Mipmapping sker automatiskt, men det finns några riktlinjer som du kan följa om du vill vara säker på att bilderna utnyttjar den här optimeringen:

- Ange egenskapen `smoothing` till `true` för Video-objektet vid videouppspelning (se klassen Video).
- För bitmappar behöver egenskapen `smoothing` inte anges till `true`, men kvalitetsförbättringarna blir mer påtagliga när bitmappar använder utjämning.
- Använd bitmappsstorlekar som är delbara med 4 eller 8 för tvådimensionella bilder (t.ex. 640 x 128, som kan förminskas enligt följande: 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1).

  För tredimensionella texturer använder du MIP-mappar där varje bild har en upplösning som är en tvåpotens (d.v.s. 2^n). Anta att huvudbilden har upplösningen 1 024 x 1 024 pixlar. Bilderna med lägre kvalitet i MIP-mappen blir då 512 x 512, 256 x 256, 128 x 128 ned till 1 x 1 pixlar för sammanlagt 11 bilder i MIP-mappen.

  Tänk på att mipmapping inte utförs för bitmappsinnehåll med udda höjd eller bredd.

Exempel på bitmapp: animerad snurrande måne

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Exemplet med den animerade snurrande månen visar tekniker för hur man arbetar med Bitmap-objekt och bitmappsbilddata (BitmapData-objekt). I exemplet skapas en animering av en snurrande sfärisk måne med en platt bild av månens yta som råbilddata. Följande tekniker visas:

- Läsa in en extern bild och använda dess råbilddata.
- Skapa animeringar genom att kopiera pixlar från olika delar av en källbild upprepade gånger.
- Skapa en bitmappsbild genom att ange pixelvärden

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfiler för den animerade snurrande månen finns i mappen Samples/SpinningMoon. Programmet består av följande filer:

Fil	Beskrivning
SpinningMoon.mxml eller SpinningMoon.fla	Huvudfilen för Flex (MXML) eller Flash (FLA).
com/example/programmingas3/moon/MoonSphere.as	En klass som utför funktionerna att läsa in, visa och animera månen.
moonMap.png	En bildfil som innehåller ett fotografi av månens yta. Bildfilen läses in och används för att skapa den animerade, snurrande månen.

Läsa in en extern bild som bitmappsdata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den första huvudsakliga uppgiften som utförs av det här exemplet är att en extern bildfil läses in. Bilden är ett foto av månens yta. Inläsningen hanteras av två metoder i klassen MoonSphere: konstruktorn `MoonSphere()` som initierar inläsningsprocessen och metoden `imageLoadComplete()` som anropas när den externa bilden har lästs in helt.

En extern bildfil läses in på ungefär samma sätt som en extern SWF-fil. I båda fallen används en instans av klassen flash.display.Loader för att utföra inläsningen. Koden i metoden `MoonSphere()` som börjar läsa in bilden är följande:

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

På den första raden deklareras en Loader-instans med namnet `imageLoader`. Det är den tredje raden som faktiskt startar inläsningen genom att anropa Loader-objektets `load()`-metod och skicka en URLRequest-instans som representerar URL:en för bilden som ska läsas in. På den andra raden ställs händelseavlyssnaren in, som ska aktiveras när bilden har lästs in. Observera att metoden `addEventListener()` inte anropas för själva Loader-instansen. I stället anropas den för Loader-objektets `contentLoaderInfo`-egenskap. Loader-instansen i sig skickar inte händelser som relaterar till innehållet som läses in. Dess `contentLoaderInfo`-egenskap innehåller däremot en referens till LoaderInfo-objektet som är associerat med innehållet som läses in i Loader-objektet (den externa bildfilen i det här fallet). Det LoaderInfo-objektet innehåller flera händelser som relaterar till förloppet och slutförandet för det externa innehållet, inklusive `complete`-händelsen (`Event.COMPLETE`) som utlöser ett anrop till metoden `imageLoadComplete()` när bilden är helt inläst.

Att starta inläsningen av den externa bilden är en viktig del av processen, men det är lika viktigt att veta vad som ska göras när inläsningen är klar. Som du kan se i koden ovan anropas funktionen `imageLoadComplete()` när bilden har lästs in. Funktionen gör flera saker med inlästa bilddata, vilket beskrivs längre fram. För att kunna använda bilddata behöver funktionen först åtkomst till dessa data. När ett Loader-objekt används för att läsa in en extern bild blir den inlästa bilden en Bitmap-instans som kopplas som ett underordnat visningsobjekt till Loader-objektet. I det här fallet är Loader-instansen tillgänglig för händelseavlyssnarmetoden som en del av händelseobjektet som skickas till metoden som en parameter. De första raderna i metoden `imageLoadComplete()` är följande:

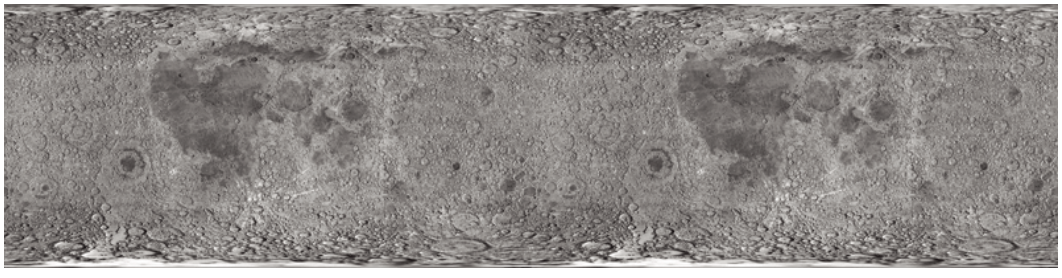
```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Observera att händelseobjektsparametern heter `event` och är en instans av klassen Event. Alla instanser av klassen Event har egenskapen `target`, som refererar till det objekt som utlöser händelsen (i det här fallet LoaderInfo-instansen som metoden `addEventListener()` anropades för, vilket beskrevs tidigare). LoaderInfo-objektet i sin tur har egenskapen `content` (så snart inläsningen är klar) som innehåller Bitmap-instansen med den inlästa bitmappsbilden. Om du vill visa bilden direkt på skärmen kan du koppla den här Bitmap-instansen (`event.target.content`) till en visningsobjektsbehållare. (Du kan också koppla objektet Loader till en visningsobjektsbehållare). I det här exemplet används emellertid det inlästa innehållet som en källa till råbilddata i stället för att visas på skärmen. Det innebär att den första raden i metoden `imageLoadComplete()` läser egenskapen `bitmapData` i den inlästa Bitmap-instansen (`event.target.content.bitmapData`) och sparar den i instansvariabeln `textureMap`, vilken används som källa till bilddata för att skapa animeringen av den roterande månen. Detta beskrivs härnäst.

Skapa animering genom att kopiera pixlar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En grundläggande definition av animering är illusionen av rörelse eller förändring som skapas av att bilden ändras över tiden. I det här exemplet är målet att skapa en illusion av att en sfärisk måne roterar runt sin lodräta axel. För den här animeringen kan du emellertid bortse från den sfäriska förvrängningsaspekten av exemplet. Titta på den faktiska bilden som läses in och används som källa till månens bilddata:



Som du ser är bilden inte en eller flera sfärer. Den är ett rektangulärt foto av månens yta. Eftersom fotot tagits exakt vid månens ekvator är de delar av bilden som är överst och underst utsträckta och förvrängda. För att bli av med förvrängningen och ge bilden ett sfäriskt utseende använder vi ett förskjutningsfilter, vilket beskrivs senare. Eftersom källbilden är en rektangel behöver koden bara förflytta månytan i vågrät riktning för att skapa en illusion av att sfären roterar.

Observera att bilden faktiskt innehåller två kopior av månytebilden bredvid varandra. Det är den här bilden som är källbilden som bilddata kopieras från flera gånger för att skapa ett intryck av rörelse. Genom att placera två kopior av bilden bredvid varandra skapas lättare en kontinuerlig, oavbruten rullningseffekt. Låt oss nu gå igenom animeringsprocessen steg för steg för att se hur detta fungerar.

I processen används egentligen två separata ActionScript-objekt. Det första är den inlästa källbilden, som är den kod som representeras av BitmapData-instansen `textureMap`. `textureMap` fylls i med bilddata så snart den externa bildfilen läses in. För detta används följande kod:

```
textureMap = event.target.content.bitmapData;
```

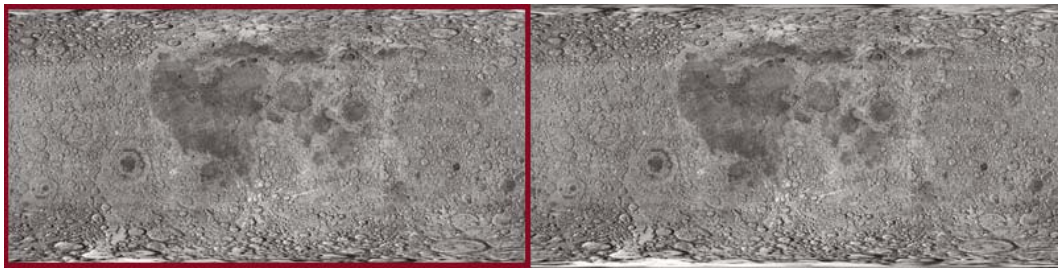
Innehållet i `textureMap` är den rektangulära månbilden. Det andra objektet som används är Bitmap-instansen `sphere`, som används för att skapa den animerade rotationen och som är det faktiska visningsobjekt som visar månbilden på skärmen. Precis som `textureMap` skapas objektet `sphere` och fylls i med sina ursprungliga bilddata i metoden `imageLoadComplete()`. För detta används följande kod:

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                        new Rectangle(0, 0, sphere.width, sphere.height),
                        new Point(0, 0));
```

Som du kan se i koden skapas en instans av `sphere`. Dess `bitmapData`-egenskap (de råbilddata som visas av `sphere`) skapas med samma höjd och halva bredden för `textureMap`. Med andra ord kommer innehållet i `sphere` att ha samma storlek som ett månfoto (eftersom bilden `textureMap` innehåller två månfoton bredvid varandra). Därefter fylls `bitmapData`-egenskapen i med bilddata med metoden `copyPixels()`. Parametrarna i anropet till metoden `copyPixels()` anger flera saker:

- Den första parametern indikerar att bilddata kopieras från `textureMap`.
- Den andra parametern, som är en ny Rectangle-instans, anger från vilken del av `textureMap` ögonblicksbilden ska tas. I det här fallet är ögonblicksbilden en rektangel som börjar i övre vänstra hörnet av `textureMap` (anges av de första två `Rectangle()`-parametrarna: `0, 0`) och rectangle-ögonblicksbildens bredd och höjd matchar egenskaperna `width` och `height` för `sphere`.
- Den tredje parametern, som är en ny Point-instans med x- och y-värdena `0`, definierar målet för pixeldata – i det här fallet det övre vänstra hörnet (0, 0) av `sphere.bitmapData`.

Visuellt sett kopierar koden pixlarna från `textureMap` som markeras i följande bild och klistrar in dem på `sphere`. Med andra ord är BitmapData-innehållet i `sphere` den del av `textureMap` som markeras här:



Kom ihåg att detta bara är utgångsläget för `sphere` – det första bildinnehållet som kopieras till `sphere`.

När källbilden har lästs in och `sphere` har skapats är den sista uppgiften som utförs av metoden `imageLoadComplete()` att ställa in animeringen. Animeringen drivs av en Timer-instans som heter `rotationTimer`. Den skapas och startas av följande kod:

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

Koden skapar först Timer-instansen `rotationTimer`. Den parameter som skickas till `Timer()`-konstruktorn anger att `rotationTimer` ska utlösa dess `timer`-händelse var femtonde millisekund. Därefter anropas metoden `addEventListener()` som anger, att när händelsen `timer` (`TimerEvent.TIMER`) inträffar, ska metoden `rotateMoon()` anropas. Slutligen startas timern genom att dess `start()`-metod anropas.

På grund av hur `rotationTimer` är definierad anropar Flash Player metoden `rotateMoon()` i klassen MoonSphere var femtonde millisekund. Det är där månen animeras. Källkoden för metoden `rotateMoon()` är följande:

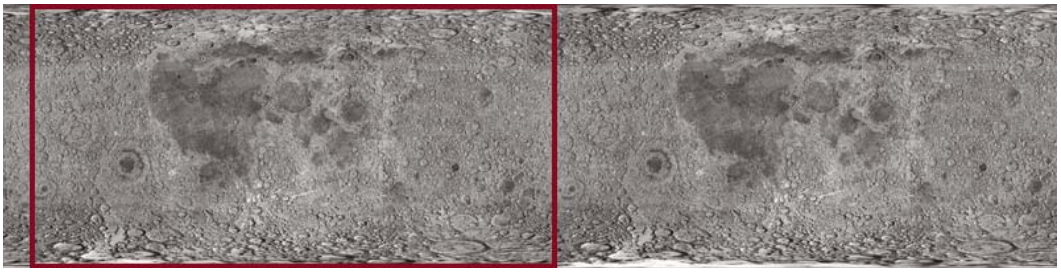
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.Data.copyPixels(textureMap,
                                    new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                    new Point(0, 0));

    event.updateAfterEvent();
}
```

Koden gör tre saker:

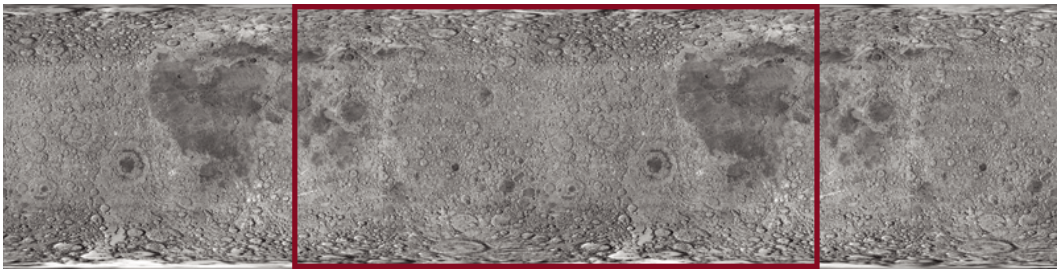
1 Värdet för variabeln `sourceX` (som ursprungligen har värdet 0) ökas med 1.

```
sourceX += 1;
```

Som du ser används `sourceX` för att avgöra var i `textureMap` pixlarna ska kopieras från till `sphere`. Den här kodens effekt är alltså att flytta rektangeln en pixel till höger i `textureMap`. Om vi går tillbaka till den visuella återgivningen kommer källrektangeln att ha flyttats flera pixlar till höger efter några animeringscykler:

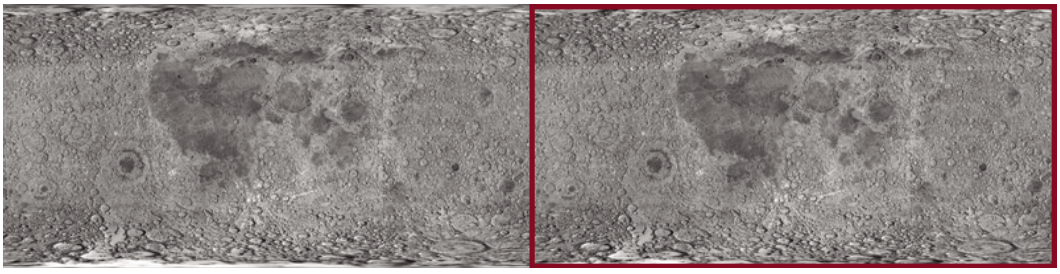


Efter ytterligare några cykler kommer rektangeln att ha rört sig ännu längre:



Denna gradvisa, konstanta förflyttning av varifrån pixlarna kopieras är nyckeln till animeringen. Genom att källplatsen sakta men säkert flyttas till höger, kommer bilden som visas på skärmen i `sphere` att se ut som om den hela tiden rör sig åt vänster. Det är därför källbilden (`textureMap`) behöver ha två kopior av fotot av månytan. Eftersom rektangeln hela tiden rör sig åt höger är den för det mesta inte över ett enda månfoto, utan överlappar båda månfotona.

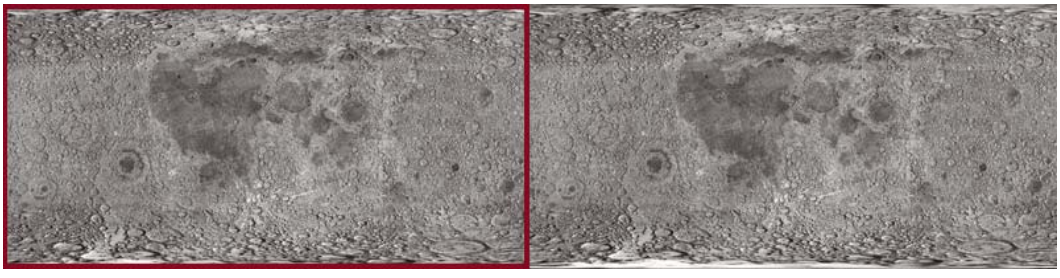
2 Det finns ett problem med att källrektangeln hela tiden rör sig åt höger. Så småningom kommer rektangeln att komma till högerkanten av `textureMap`. Då kommer det inte att finnas fler månpixlar att kopiera till `sphere`:



Nästa kodrader åtgärdar detta problem:

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

Koden kontrollerar om `sourceX` (rektangelns vänstra kant) har nått mitten av `textureMap`. I så fall återställs `sourceX` till 0 så att den flyttas tillbaka till vänstra kanten av `textureMap` och börjar om cykeln:



3 När rätt värde för `sourceX` har beräknats är det sista steget för animeringen att faktiskt kopiera den nya källrektangelns pixlar till `sphere`. Koden som gör detta liknar koden som ursprungligen fanns i `sphere` (beskrivs ovan) väldigt mycket. Den enda skillnaden är att i det här fallet placeras rektangelns vänsterkant vid `sourceX` i konstruktoranropet `new Rectangle()`:

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Kom ihåg att den här koden anropas om och om igen var femtonde millisekund. Eftersom källrektangelns plats flyttas hela tiden och pixlarna kopieras till `sphere`, blir effekten på skärmen att månfotot som representeras av `sphere` hela tiden rör sig. Med andra ord ser månen ut att rotera oavbrutet.

Skapa ett sfäriskt utseende.

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Månen är förstås en sfär och inte en rektangel. Därför måste exemplet ta det rektangulära fotot av månytan som animeras kontinuerligt och konvertera det till en sfär. Detta görs i två steg: en mask används för att dölja allt innehåll förutom ett cirkelformat område av månytefotot, och ett förskjutningsfilter används för att förvränga månfotots utseende så att det ser tredimensionellt ut.

Först används en cirkelformad mask för att dölja allt innehåll i MoonSphere-objektet förutom den sfär som skapas av filtret. I följande kod skapas en mask som en Shape-instans. Den tillämpas som mask för MoonSphere-instansen:

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Observera, att eftersom MoonSphere är ett visningsobjekt (det är baserat på klassen Sprite) kan masken tillämpas direkt på MoonSphere-instansen med den ärvda egenskapen `mask`.

Att bara dölja delar av fotot med en cirkelformad mask räcker inte för att skapa en realistisk effekt av en roterande sfär. På grund av det sätt som fotot av månens yta tagits är inte dess dimensioner proportionerliga. De delar av bilden som är högst upp och längst ned är mer förvrängda och utsträckta jämfört med delarna nära ekvatorn. För att månfotot ska se tredimensionellt ut förvränger vi det med ett förskjutningsfilter.

Ett förskjutningsfilter är ett filter som används för att förvränga en bild. I det här fallet förvrängs fotot så att det ser mer realistiskt ut genom att bildens övre och nedre del trycks ihop i vågrät riktning medan mittpartiet inte ändras. Förutsatt att filtret används på en fyrkantig del av fotot, kommer det att bli en cirkel när övre och nedre delen kläms ihop utan att mittpartiet påverkas. En sidoeffekt av animeringen av den här förvrängda bilden blir att mitten av bilden verkar röra sig längre i faktiskt pixelavstånd än de övre och nedre delarna, vilket ger intrycket av att cirkeln faktiskt är ett tredimensionellt objekt (en sfär).

I följande kod skapas förskjutningsfiltret `displaceFilter`:

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

Den första parametern `fisheyeLens` kallas för mappningsbilden. I det här fallet är den ett BitmapData-objekt som skapas programmatiskt. Hur den bilden skapas beskrivs i avsnittet ”Skapa en bitmappsbild genom att ange pixelvärden” på sidan 262. Den andra parametern anger vid vilken position i den filtrerade bilden som filtret ska tillämpas, vilka färgkanaler som ska användas för att styra förskjutningseffekten och i vilken utsträckning de ska påverka förskjutningen. När förskjutningsfiltret har skapats tillämpas det på `sphere`, fortfarande inom metoden `imageLoadComplete()`:

```
sphere.filters = [displaceFilter];
```

Den färdiga bilden med både mask och förskjutningsfilter tillämpade ser ut så här:

För varje cykel i animeringen av den roterande månen skrivs BitmapData-innehållet i sphere över av en ny ögonblicksbild av källbilden. Filtret behöver däremot inte tillämpas igen varje gång. Det beror på att filtret tillämpas på Bitmap-instansen (visningsobjektet) i stället för på bitmappsdata (råa pixeldata). Kom ihåg att Bitmap-instansen egentligen inte är faktiska bitmappsdata, den är ett visningsobjekt som visar bitmappsdata på skärmen. Man skulle kunna säga att en Bitmap-instans är som projektorn som används för att visa diabilder på en duk, och att BitmapData-objektet är den diabild som visas av projektorn. Ett filter kan tillämpas direkt på ett BitmapData-objekt, vilket skulle motsvara att rita direkt på en diabild för att ändra bilden. Ett filter kan också tillämpas på visningsobjekt, till exempel en Bitmap-instans. Det skulle motsvara att placera ett filter framför projektorns lins för att förvränga den bild som visas på skärmen (utan att ändra diabilden i sig). Eftersom råa bitmappsdata är tillgängliga via bitmapData-egenskapen för en Bitmap-instans, skulle man ha kunnat tillämpa filtret direkt på dessa data. I det här fallet är det dock bättre att tillämpa filtret på Bitmap-visningsobjektet i stället för på bitmappsdata.

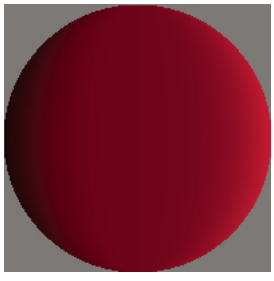
Mer information om hur du använder förskjutningsfiltret i ActionScript finns i ”Filtrera visningsobjekt” på sidan 268.

Skapa en bitmappsbild genom att ange pixelvärden

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En viktig aspekt av förskjutningsfiltret är att det faktiskt används två bilder i det. Den ena bilden (källbilden) är den bild som faktiskt ändras av filtret. I det här exemplet är källbilden Bitmap-instansen `sphere`. Den andra bilden som används av filtret kallas mappningsbilden. Mappningsbilden visas aldrig på skärmen. I stället används färgen för var och en av dess pixlar som indata till förskjutningsfunktionen. Pixelns färg vid en viss x, y-koordinat i mappningsbilden avgör hur mycket förskjutning (fysisk lägesändring) som ska tillämpas på pixeln vid den x, y-koordinaten i källbilden.

För att ett förskjutningsfilter ska kunna användas för att skapa en sfäreffekt måste alltså rätt mappningsbild finnas i exemplet. Den ska ha grå bakgrund och en cirkel som är ifylld med en övertoning av en enda färg (röd) som övergår från mörk till ljus i vågrät riktning, vilket visas här:



Eftersom bara en mappningsbild och ett filter används i exemplet skapas bara mappningsbilden en gång, i metoden `imageLoadComplete()` (d.v.s. när den externa bilden har lästs in). Mappningsbilden, som heter `fisheyeLens`, skapas genom att metoden `createFisheyeMap()` i klassen MoonSphere anropas:

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Inuti metoden `createFisheyeMap()` ritas mappningsbilden en pixel i taget med metoden `setPixel()` i klassen BitmapData. Den fullständiga koden för metoden `createFisheyeMap()` listas här, följd av en steg-för-steg-diskussion om hur den fungerar:

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

När metoden först anropas får den parametern `radius` som anger radien för den cirkelformade bild som ska skapas. Sedan skapar koden det BitmapData-objekt som cirkeln ska ritas på. Det objektet, som heter `result`, skickas sedan tillbaka som returvärde för metoden. Som du kan se i följande kodavsnitt skapas BitmapData-instansen `result` med en bredd och höjd som är lika med cirkelns diameter, utan genomskinlighet (den tredje parametern är `false`) och i förväg ifylld med färgen `0x808080` (mellangrå):

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

Sedan används två slingor för att iterera genom alla pixlar i bilden. Den yttre slingan går igenom varje kolumn i bilden från vänster till höger (variabeln `i` representerar den vågräta positionen för den pixel som manipuleras). Den inre slingan går igenom varje pixel i den aktuella kolumnen uppifrån och ner (variabeln `j` representerar den lodräta positionen för den pixel som manipuleras). Koden för slingorna (den inre slingans innehåll har utelämnats) visas här:

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

När slingan går igenom pixlarna en i taget beräknas ett värde för varje pixel (färgvärdet för den pixeln i mappningsbilden). Detta görs i fyra steg:

1. Den aktuella pixelns avstånd från cirkelns mitt längs x-axeln `(i - radius)` beräknas. Detta värde divideras med radien så att det blir en procentandel av radien i stället för ett absolut avstånd (`(i - radius) / radius`). Procentvärdet sparas i en variabel som heter `pctX` och motsvarande värde för y-axeln beräknas och sparas i variabeln `pctY`, vilket visas i följande kod:

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2. Med en vanlig trigonometrisk formel (Pythagoras sats) beräknas det linjära avståndet mellan cirkelns mitt och den aktuella punkten från värdena `pctX` och `pctY`. Det värdet sparas i en variabel som heter `pctDistance`, vilket visas här:

   ```
   var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
   ```

3. Sedan kontrolleras om avståndet är mindre än 1 (vilket innebär 100 % av radien, med andra ord om pixeln är inom cirkelns radie eller inte). Om pixeln är inuti cirkeln tilldelas den ett beräknat färgvärde (utelämnas här, men beskrivs i steg 4). Om pixeln är utanför cirkeln händer inget mer med den pixeln och dess färg förblir mellangrå:

   ```
   if (pctDistance < 1)
   {
       ...
   }
   ```

4. För pixlar som är inuti cirkeln beräknas ett färgvärde. Den slutliga färgen blir en röd nyans som ligger mellan svart (0 % röd) längst till vänster i cirkeln och klarröd (100 % röd) längst till höger i cirkeln. Färgvärdet beräknas ursprungligen i tre delar (röd, grön och blå) vilket visas här:

   ```
   red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
   green = 0;
   blue = 0;
   ```

Observera att bara den röda delen av färgen (variabeln `red`) faktiskt har ett värde. De gröna och blå värdena (variablerna `green` och `blue`) visas här av tydlighetsskäl, men de kan utelämnas. Eftersom syftet med den här metoden är att skapa en cirkel som innehåller en röd övertoning behövs inga gröna eller blå värden.

När de tre enskilda färgvärdena har bestämts kombineras de till ett enda heltalsfärgvärde med en standardalgoritm för bitflyttning, som visas i följande kod:

```
rgb = (red << 16 | green << 8 | blue);
```

När färgvärdet slutligen har beräknats kan det tilldelas den aktuella pixeln med metoden `setPixel()` i BitmapData-objektet `result`, vilket visas här:

```
result.setPixel(i, j, rgb);
```

Asynkron avkodning av bitmappsbilder

Flash Player 11 och senare, Adobe AIR 2.6 och senare

När du arbetar med bitmappsbilder kan du avkoda och läsa in dem asynkront för att förbättra programmets upplevda prestanda. I många fall tar det lika lång tid att avkoda en bitmapp asynkront som synkront. Men bitmappsbilden avkodas i en separat tråd innan det associerade `Loader`-objektet skickar `COMPLETE`-händelsen. Du kan således avkoda större bilder asynkront efter att de lästs in.

Med klassen `ImageDecodingPolicy` i paketet `flash.system` kan du ange inläsningsschema för bitmappar. Standardinläsningschemat är synkront.

Avkodningsprincip för bitmappar	Inläsningsschema för bitmappar	Beskrivning
`ImageDecodingPolicy.ON_DEMAND`	Synkron	Inlästa bilder avkodas vid åtkomst av bilddata. Använd den här principen för att avkoda mindre bilder. Du kan också använda den här principen om programmet inte innehåller komplexa effekter och övergångar.
`ImageDecodingPolicy.ON_LOAD`	Asynkron	Inlästa bilder avkodas vid inläsning, innan händelsen `COMPLETE` skickas. Praktiskt för större bilder (större än 10 MP). När du utvecklar AIR-baserade mobilprogram med sidövergångar använder du den här inläsningsprincipen för bitmappar för att förbättra programmets upplevda prestanda.

Obs! *Om den fil som läses in är en bitmappsbild, och avkodningsprincipen är `ON_LOAD`, avkodas bilden asynkront innan `COMPLETE`-händelsen skickas.*

Följande kod visar hur klassen `ImageDecodingPolicy` används:

```
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD
var loader:Loader = new Loader();
loader.load(new URLRequest("http://www.adobe.com/myimage.png"), loaderContext);
```

Du kan fortfarande använda `ON_DEMAND`-avkodning med metoderna `Loader.load()` och `Loader.loadBytes()`. Alla andra metoder som får ett `LoaderContext`-objekt som argument kommer dock att ignorera alla `ImageDecodingPolicy`-värden som skickas.

Följande exempel visar skillnaden mellan att avkoda en bitmappsbild asynkront och synkront:

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.ImageDecodingPolicy;
    import flash.system.LoaderContext;

    public class AsyncTest extends Sprite
    {
        private var loaderContext:LoaderContext;
        private var loader:Loader;
        private var urlRequest:URLRequest;
        public function AsyncTest()
        {
            //Load the image synchronously
            loaderContext = new LoaderContext();
            //Default behavior.
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_DEMAND;
            loader = new Loader();
            loadImageSync();

            //Load the image asynchronously
            loaderContext = new LoaderContext();
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;
            loader = new Loader();
            loadImageASync();
        }

        private function loadImageASync():void{
            trace("Loading image asynchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onAsyncLoadComplete);
```

```
        }

        private function onAsyncLoadComplete(event:Event):void{
            trace("Async. Image Load Complete");
        }

        private function loadImageSync():void{
            trace("Loading image synchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onSyncLoadComplete);
        }

        private function onSyncLoadComplete(event:Event):void{
            trace("Sync. Image Load Complete");
        }
    }
}
```

Om du vill se en demonstration av effekten av olika avkodningsprinciper ska du besöka Thibaud Imbert: Asynchronous bitmap decoding in the Adobe Flash runtimes

Kapitel 14: Filtrera visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Historiskt sett är tillämpning av filtereffekter på bitmappsbilder något som görs med specialiserade bildredigeringsprogram, som Adobe Photoshop® och Adobe Fireworks®. ActionScript 3.0 innehåller flash.filters-paketet med en rad filterklasser för bitmappeffekter. Med de här effekterna kan utvecklare lägga till filter för bitmappar och visningsobjekt i programmet och på så sätt uppnå många av de effekter som är tillgängliga i bildhanteringsprogram.

Grunderna för filtrering av visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett sätt att förbättra ett program är att lägga till enkla grafikeffekter. Du kan lägga till en skugga bakom ett foto för att skapa ett tredimensionellt utseende eller glöd runt en knapp för att visa att den är aktiv. I ActionScript 3.0 finns tio filter, som kan tillämpas på alla visningsobjekt och på BitmapData-instanser. De inbyggda filtren omfattar både grundläggande varianter, som skugga och glöd, och mer avancerade filter, som förskjutningskarta och faltning.

Obs! *Utöver de inbyggda filtren kan du även programmera egna filter och effekter med Pixel Bender. Läs mer i* *"Arbeta med Pixel Bender-skuggningar" på sidan 301.*

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kan stöta på när du skapar filter:

Avfasning En kant som skapas genom att pixlarna på två sidor görs ljusare och pixlarna på motsatta två sidor görs mörkare. Med den här effekten ser kanten ut att vara tredimensionell. Effekten används ofta för upphöjda eller nedtryckta knappar och liknande bilder.

Faltning Pixlarna i en bild förvrängs genom att varje pixels värde kombineras med värdet för några eller alla intilliggande pixlar, med olika proportioner.

Förskjutning Pixlarna i en bild förskjuts eller flyttas till en ny position.

Matris Ett rutnät med tal som används för att utföra vissa matematiska beräkningar genom att talen i rutnätet tillämpas på olika värden och resultatet kombineras.

Fler hjälpavsnitt

flash.filters-paketet

flash.display.DisplayObject.filters

flash.display.BitmapData.applyFilter()

Skapa och tillämpa filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med filter kan du tillämpa en mängd effekter på bitmappsbilder och visningsobjekt, från skuggor till avfasning och oskärpa. Varje filter definieras som en klass. När du tillämpar filter måste du därför skapa instanser av filterobjekt på samma sätt som när du skapar andra objekt. När du har skapat en instans av ett filterobjekt kan det enkelt tillämpas på ett visningsobjekt med objektets `filters`-egenskap, eller för ett BitmapData-objekt, med metoden `applyFilter()`.

Skapa ett filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill skapa ett filterobjekt anropar du bara konstruktormetoden hos den filterklass du har valt. Om du till exempel vill skapa ett DropShadowFilter-objekt använder du följande kod:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Konstruktorn `DropShadowFilter()` visas inte här, men den (liksom alla filterklassers konstruktorer) tar flera valfria parametrar som kan användas för att anpassa filtereffektens utseende.

Tillämpa ett filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du har skapat ett filterobjekt med konstruktorn kan du tillämpa det på ett visningsobjekt eller ett BitmapData-objekt. Hur du tillämpar filtret beror på vilket objekt du tillämpar det på.

Tillämpa ett filter på ett visningsobjekt

När du tillämpar filtereffekter på ett visningsobjekt gör du det med egenskapen `filters`. Egenskapen `filters` i ett visningsobjekt är en Array-instans vars element är de filterobjekt som tillämpas på visningsobjektet. Om du vill tillämpa ett enda filter på ett visningsobjekt skapar du filterinstansen, lägger till den i en Array-instans och tilldelar Array-objektet till visningsobjektets `filters`-egenskap:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Om du vill tilldela flera filter till objektet lägger du bara till dem i Array-instansen innan du tilldelar den till egenskapen `filters`. Du kan lägga till flera objekt i en Array genom att skicka dem som parametrar till dess konstruktor. I följande kod tillämpas till exempel ett avfasningsfilter och ett glödfilter på det visningsobjekt som skapats tidigare:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

När du skapar arrayen som innehåller filtren kan du skapa den med konstruktorn `new Array()` (vilket visas i de föregående exemplen) eller också kan du använda litteral Array-syntax och placera filtren inom hakparenteser (`[]`). Till exempel gör följande kodrad:

```
var filters:Array = new Array(dropShadow, blur);
```

samma sak som den här kodraden:

```
var filters:Array = [dropShadow, blur];
```

Om du tillämpar flera filter på visningsobjekt tillämpas de kumulativt ett i taget. Om en filterarray till exempel har två element, ett avfasningsfilter som läggs till först och ett skuggfilter som läggs till därefter, tillämpas skuggfiltret på både avfasningsfiltret och visningsobjektet. Detta beror på att skuggfiltret kommer efter avfasningsfiltret i filterarrayen. Om du inte vill tillämpa filtren kumulativt måste du tillämpa varje filter på en ny kopia av visningsobjektet.

Om du bara tilldelar ett eller ett fåtal filter till ett visningsobjekt kan du skapa filterinstansen och tilldela den till objektet med en enda programsats. Följande kodrad tillämpar till exempel ett oskärpefilter på ett visningsobjekt med namnet `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

I ovanstående kod skapas en Array-instans med litteral Array-syntax (hakparenteser), en BlurFilter-instans skapas som ett element i arrayen och arrayen tilldelas till egenskapen `filters` i visningsobjektet `myDisplayObject`.

Ta bort filter från ett visningsobjekt

Att ta bort alla filter från ett visningsobjekt är enkelt. Tilldela bara värdet null till egenskapen `filters`:

```
myDisplayObject.filters = null;
```

Om du har tillämpat flera filter på ett objekt och bara vill ta bort ett av dem måste du gå igenom flera steg för att ändra `filters`-egenskapens array. Mer information finns i ”Potentiella problem vid arbete med filter” på sidan 271.

Tillämpa ett filter på ett BitmapData-objekt

För att kunna tillämpa ett filter på ett BitmapData-objekt måste du använda BitmapData-objektets `applyFilter()`-metod:

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

Metoden `applyFilter()` tillämpar ett filter på ett BitmapData-källobjekt, vilket ger en ny, filtrerad bild. Metoden ändrar inte originalkällbilden. I stället gör filtret att källbilden sparas i BitmapData-instansen, som metoden `applyFilter()` sedan anropas för.

Så fungerar filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Filtrering av visningsobjekt fungerar genom att en kopia av originalobjektet sparas i cache-minnet som en genomskinlig bitmappsbild.

När ett filter har använts på ett visningsobjekt cachelagras objektet som en bitmapp så länge objektet har en giltig filterlista. Källbitmappsbilden används sedan som originalbild för alla efterföljande filtereffekter som tillämpas.

Varje visningsobjekt har vanligtvis två bitmappsbilder: en med det ursprungliga ofiltrerade källvisningsobjektet och en för den slutliga bilden efter filtrering. Den slutliga bilden används för rendering. Så länge visningsobjektet inte ändras behöver den slutgiltiga bilden inte uppdateras.

Potentiella problem vid arbete med filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns flera saker som kan skapa förvirring eller problem och som du bör tänka på när du arbetar med filter.

Filter och bitmappscache-lagring

För att ett filter ska kunna tillämpas på ett visningsobjekt måste bitmappscache-lagring vara aktiverad för det objektet. När du använder ett filter på ett visningsobjekt vars `cacheAsBitmap`-egenskap är inställd på `false` anges objektets `cacheAsBitmap`-egenskap automatiskt till `true`. Om du senare tar bort alla filter från visningsobjektet återställs egenskapen `cacheAsBitmap` till sitt senaste värde.

Ändra filter vid körning

Om ett eller flera filter redan tillämpas på ett visningsobjekt kan du inte ändra uppsättningen filter genom att lägga till fler filter eller ta bort filter från `filters`-egenskapens array. Om du vill lägga till eller ändra uppsättningen filter som tillämpas måste du i stället göra ändringarna i en separat array, och sedan tilldela denna array till visningsobjektets filters-egenskap så att filtren tillämpas på objektet. Det enklaste sättet att göra detta är att läsa in `filters`-egenskapens array till en Array-variabel och göra ändringarna i denna tillfälliga array. Sedan tilldelar du om denna array tillbaka till visningsobjektets `filters`-egenskap. I mer komplicerade fall kanske du behöver ha en separat huvudarray med filter. Du kan göra ändringarna i denna huvudfilterarray, och sedan tilldela om huvudarrayen till visningsobjektets `filters`-egenskap efter varje ändring.

Lägga till fler filter

I följande kod visas hur du lägger till ett till filter för ett visningsobjekt där ett eller flera filter redan tillämpas. Först tillämpas ett glödfilter på visningsobjektet `myDisplayObject`. När användaren sedan klickar på visningsobjektet anropas funktionen `addFilters()`. I den här funktionen läggs ytterligare två filter till i `myDisplayObject`:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

Ta bort ett filter från en uppsättning filter

Om flera filter tillämpas på ett visningsobjekt, och du vill ta bort ett av filtren medan de andra fortfarande ska användas, kopierar du filtren till en tillfällig array, tar bort filtret du inte vill ha från arrayen och tilldelar sedan om den tillfälliga arrayen tillbaka till visningsobjektets `filters`-egenskap. Olika sätt att ta bort ett eller flera element från en array beskrivs i ”Hämta värden och ta bort arrayelement” på sidan 30.

Det enklaste är att ta bort det översta filtret för objektet (det som tillämpats senast på objektet). Du tar bort filtret från arrayen med klassens Arrays `pop()`-metod:

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

När du tar bort det understa filtret (det första som tillämpats på objektet) använder du samma kod, men använder klassen Arrays `shift()`-metod i stället för metoden `pop()`.

Om du vill ta bort ett filter från mitten av en array med filter (förutsatt att arrayen har fler än två filter) kan du använda metoden `splice()`. Du måste känna till indexet (placeringen i arrayen) för filtret som du vill ta bort. Följande kod tar till exempel bort det andra filtret (filtret vid index 1) från ett visningsobjekt:

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Ta reda på ett filters index

Du måste veta vilket filter som du vill ta bort från arrayen, så att du vet filtrets index. Du måste antingen känna till (genom hur programmet är designat), eller beräkna indexet för filtret som ska tas bort.

Det bästa är att designa programmet så att filtret som du vill ta bort alltid finns på samma plats i uppsättningen med filter. Om du till exempel har ett enda visningsobjekt med ett faltningsfilter och ett skuggfilter (i den ordningen), och du vill ta bort skuggfiltret men behålla faltningsfiltret, finns filtret i en känd position (det översta filtret) så att du i förväg vet vilken Array-metod som ska användas (i det här fallet `Array.pop()` för att ta bort skuggfiltret).

Om filtret som du vill ta bort alltid är av en viss typ, men inte nödvändigtvis finns i samma position i uppsättningen med filter, kan du kontrollera datatypen för filtren i arrayen när du vill ta reda på vilket som ska tas bort. Följande kod tar till exempel reda på vilket filter i en uppsättning som är ett glödfilter, och tar bort det filtret från uppsättningen.

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

I ett mer komplicerat fall, till exempel om filtret som ska tas bort väljs under körning, är det bäst att ha en separat, beständig kopia av filterarrayen som fungerar som huvudlista för filtren. När du vill ändra i uppsättningen med filter ändrar du huvudlistan och tillämpar sedan den filterarrayen som visningsobjektets `filters`-egenskap.

I följande kodexempel tillämpas flera faltningsfilter på ett visningsobjekt för att skapa olika visuella effekter. Vid en senare punkt i programmet tas ett av dessa filter bort medan de andra finns kvar. I det här fallet behåller koden både en huvudkopia av filterarrayen och en referens till filtret som ska tas bort. Att söka efter och ta bort ett speciellt filter påminner om det tidigare tillvägagångssättet, men i stället för att skapa en tillfällig kopia av filterarrayen ändras huvudkopian och den tillämpas sedan på visningsobjektet.

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

I det här fallet (när du jämför en lagrad filterreferens med objekten i filterarrayen för att ta reda på vilket filter som ska tas bort), *måste* du behålla en separat kopia av filterarrayen. Koden fungerar inte om du jämför den lagrade filterreferensen med elementen i en tillfällig array som kopieras från visningsobjektets `filters`-egenskap. Detta har interna orsaker. När du tilldelar en array till egenskapen `filters` skapas automatiskt en kopia av varje filterobjekt i arrayen. Dessa kopior (och inte de ursprungliga objekten) tillämpas på visningsobjektet. När du läser in egenskapen `filters` till en tillfällig array innehåller den tillfälliga arrayen referenser till de kopierade filterobjekten i stället för referenser till de ursprungliga filterobjekten. Om du i föregående exempel försöker att ta reda på indexet för `filterToRemove` genom att jämföra det med filtren i en tillfällig filterarray, hittas därför inga matchningar.

Filter och objektomformningar

Inget filterområde (till exempel skugga) utanför visningsobjektets begränsningsram anses vara del av ytan när det gäller att identifiera en träff (avgöra om en instans överlappar eller korsar en annan instans). Eftersom klassen DisplayObjects träffidentifiering är vektorbaserad, kan den inte utföras på ett bitmappsresultat. Exempel: Om du använder ett avfasningsfilter på en knappinstans, är träffidentifieringen inte tillgänglig på avfasningsdelen i instansen.

Skalning, rotering och skevning stöds inte av filter. Om själva visningsobjektet som filtreras är skalat (om `scaleX` och `scaleY` inte är 100 %), skalas inte filtereffekten med instansen. Det betyder att den ursprungliga formen i instansen roteras, skalas eller skevas. Däremot roteras, skalas eller skevas inte filtret med instansen.

Du kan animera en instans med ett filter för att skapa realistiska effekter eller kapsla in instanser och använda klassen BitmapData för att animera filter och få den här effekten.

Filter och bitmappsobjekt

Om du använder ett filter på ett BitmapData-objekt ställs egenskapen `cacheAsBitmap` automatiskt in på `true`. Det innebär att filtret i själva verket tillämpas på kopian av objektet i stället för på originalet.

Kopian placeras sedan på huvudvisningsobjektet (över det ursprungliga objektet) så nära närmsta pixel som möjligt. Om den ursprungliga bitmappens gränser ändras, skapas den filtrerade kopian av bitmappen om från originalet i stället för att sträckas ut eller förvrängas.

Om du tar bort alla filter för ett visningsobjekt återställs egenskapen `cacheAsBitmap` till den inställning den hade innan filtret tillämpades.

Tillgängliga visningsfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 finns tio filterklasser som kan tillämpas på visningsobjekt och på BitmapData-objekt:

- Avfasningsfilter (klassen BevelFilter)
- Oskärpefilter (klassen BlurFilter)
- Skuggfilter (klassen DropShadowFilter)
- Glödfilter (klassen GlowFilter)
- Övertoningsavfasningsfilter (klassen GradientBevelFilter)
- Övertoningsglödfilter (klassen GradientGlowFilter)
- Färgmatrisfilter (klassen ColorMatrixFilter)
- Faltningsfilter (klassen ConvolutionFilter)
- Förskjutningsfilter (klassen DisplacementMapFilter)
- Skuggningsfilter (klassen ShaderFilter)

De sex första filtren är enkla filter som kan användas för att skapa en viss effekt och för att anpassa effekten till viss del. De sex filtren kan tillämpas med ActionScript och kan dessutom tillämpas på objekt i Flash Professional med filterpanelen. Det innebär att även om du tillämpar filter med ActionScript kan du, om du har Flash Professional, använda det visuella gränssnittet för att snabbt pröva olika filter och inställningar för att se hur du ska få den effekt du vill ha.

De sista fyra filtren är bara tillgängliga i ActionScript. Dessa filter, färgmatrisfiltret, faltningsfiltret, förskjutningsfiltret och skuggningsfiltret, är mer flexibla än de typer av effekter som de kan användas för att skapa. De är inte optimerade för en enda effekt, utan har större kraft och flexibilitet. Till exempel kan du använda faltningsfiltret för att skapa effekter som oskärpa, relief, skärpa och omformningar samt för att hitta färgkanter genom att välja olika värden för dess matris.

Alla filter, oavsett om de är enkla eller komplicerade, kan anpassas med hjälp av deras egenskaper. I allmänhet finns det två sätt att ställa in filteregenskaper. För alla filter kan du ställa in egenskaperna genom att ange parametervärden i filterobjektets konstruktor. Alternativt kan du justera filtren senare genom att ange värden för filterobjektens egenskaper, oavsett om du ställer in filteregenskaperna genom att skicka parametrar eller inte. I de flesta kodexemplen ställs egenskaperna in direkt, eftersom det gör exemplen lättare att följa. Du kan oftast få samma resultat med färre rader kod genom att skicka värdena som parametrar i filterobjektets konstruktor. Mer information om olika filter, deras egenskaper och konstruktorparametrar finns i flash.filters-paketet i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Avfasningsfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen BevelFilter kan du lägga till en 3D-fasad kant i det filtrerade objektet. Filtret gör att objektets skarpa hörn eller kanter ser ut som om de mejslats eller fasats bort.

Med egenskaperna i klassen BevelFilter kan du anpassa avfasningens utseende. Du kan ställa in högdager- och skuggfärg, oskärpa för avfasningskanter, avfasningsvinklar och placering av avfasningskanter. Du kan även skapa en blockeringseffekt.

I följande exempel läses en extern bild in och ett avfasningsfilter tillämpas på den.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
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);

// Create the bevel filter and set filter properties.
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;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

Oskärpefilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen BlurFilter gör ett visningsobjekt och dess innehåll suddigt eller oskarpt. Oskärpeeffekter används för att ge intryck av att ett objekt inte är i fokus eller för att simulera snabba rörelser, som i rörelseoskärpa. Genom att ställa in oskärpefiltrets `quality`-egenskap på low kan du simulera en mjuk, ofokuserad linseffekt. Om du ställer in `quality`-egenskapen på high blir effekten en mjuk oskärpa som liknar gaussisk oskärpa.

I följande exempel skapas ett cirkelobjekt med metoden `drawCircle()` i klassen Graphics och ett oskärpefilter tillämpas på det:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

Skuggfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Skuggor ger intryck av att det finns en separat ljuskälla ovanför målobjektet. Ljuskällans placering och intensitet kan ändras för att ge ett antal olika skuggeffekter.

Klassen DropShadowFilter använder en algoritm som påminner som oskärpefiltrets algoritm. Den huvudsakliga skillnaden är att det finns några egenskaper till i skuggfiltret som du kan ändra för att simulera olika ljuskälleattribut (till exempel alfa, färg, förskjutning och intensitet).

Skuggfiltret gör också att du kan tillämpa anpassade omformningsalternativ på skuggans format, till exempel inre och yttre skugga och blockeringsläge.

I följande kod skapas en fyrkantig sprite och ett skuggfilter används på den:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
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);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

Glödfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen GlowFilter tillämpar en ljussättningseffekt på visningsobjekt för att efterlikna en ljuskälla som är riktad mot objektet underifrån så att en mjuk glöd uppstår.

Glödfiltret liknar skuggfiltret och innehåller egenskaper som kan användas för att ändra avståndet, vinkeln och färgen för ljuskällan för att ge olika effekter. Det finns flera alternativ för stilen på glöden i klassen GlowFilter, inklusive inre eller yttre glöd och blockeringsläge.

I följande kod skapas ett kors med klassen Sprite och ett glödfilter används på det:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
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);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

Övertoningsavfasningsfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen GradientBevelFilter kan du använda en förbättrad avfasningseffekt på visningsobjekt eller BitmapData-objekt. Om du använder en övertoningsfärg på avfasningen förbättras dess djup så att kanterna blir mer realistiska och tredimensionella.

I följande kod skapas ett rektangelobjekt med metoden `drawRect()` i klassen Shape och ett övertoningsavfasningsfilter tillämpas på det:

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
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;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

Övertoningsglödfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen GradientGlowFilter kan du använda en förbättrad glödeffekt på visningsobjekt eller BitmapData-objekt. Den här effekten ger dig mer kontroll över glödens färg och ger en mer realistisk glödeffekt. Dessutom gör övertoningsglödfiltret att du kan tillämpa en övertoningsglöd på ett objekts inre, yttre eller övre kanter.

I följande exempel ritas en cirkel på scenen och ett övertoningsglödfilter används på den. När du flyttar muspekaren till höger och nedåt ökar oskärpan i vågrät respektive lodrät riktning. Dessutom ökar oskärpans styrka varje gång du klickar på scenen.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
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();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Define a gradient glow.
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;

// Define functions to listen for two events.
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);
```

Exempel: Kombinera enkla filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande kodexempel används flera enkla filter som kombineras med en timer som gör att åtgärderna upprepas, vilket ger intryck av ett animerat trafikljus.

```
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; // opposite of 45 degrees
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;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
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();

// Add the graphics to the display list.
addChild(box);
```

```
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees,
colors, alphas, ratios, blurX, blurY, strength, quality, type, knockout);
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
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);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
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++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

Färgmatrisfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen ColorMatrixFilter används för att manipulera färg- och alfavärdena för det filtrerade objektet. Det gör att du kan skapa mättnadsändringar, nyansrotation (flytta en palett från en uppsättning färger till en annan), luminans-till-alfa-ändringar och andra färgförändringseffekter med värden från en färgkanal och också tillämpa dem på andra kanaler.

Filtret går igenom källbildens pixlar en i taget och delar upp dem i komponenterna röd, grön, blå och alfa. Sedan multipliceras värdena som anges i färgmatrisen med vart och ett av dessa värden. Resultaten läggs ihop för bestämning av vilket färgvärde som ska visas på skärmen för den pixeln. Filtrets `matrix`-egenskap är en array med 20 tal som används för att beräkna den slutgiltiga färgen. Mer information om den algoritm som används för att beräkna färgvärden finns i beskrivningen av egenskapen `matrix` för klassen ColorMatrixFilter i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Faltningsfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen ConvolutionFilter kan användas för att tillämpa en mängd bildomformningar på BitmapData-objekt eller visningsobjekt, till exempel oskärpa, kantidentifiering, skärpa, relief och avfasning.

Faltningsfiltret går igenom varje pixel i källbilden en i taget och avgör dess slutgiltiga färg genom att använda värdet för pixeln och de omgivande pixlarna. En matris, som specificeras som en array med numeriska värden, anger i vilken utsträckning värdet för varje närliggande pixel påverkar det slutgiltiga värdet.

Ta till exempel den vanligaste matristypen, d.v.s. en 3 x 3-matris. Matrisen innehåller nio värden:

```
N   N   N
N   P   N
N   N   N
```

När faltningsfiltret används på en viss pixel läser filtret pixelns eget färgvärde (”P” i exemplet) och även värdena för de omgivande pixlarna (”N” i exemplet). Genom att ställa in värdena i matrisen anger du dock hur hög prioritet vissa pixlar har när det gäller att påverka bilden.

I följande matris, som tillämpas med ett faltningsfilter, lämnas bilden oförändrad:

```
0   0   0
0   1   0
0   0   0
```

Orsaken till att bilden är oförändrad är att den ursprungliga pixelns värde har den relativa styrkan 1 när det gäller att avgöra den slutgiltiga pixelns färg, medan de omgivande pixlarnas värden har den relativa styrkan 0. Det innebär att deras färger inte påverkar den slutgiltiga bilden.

På samma sätt gör den här matrisen att pixlarna i en bild flyttas en pixel till vänster:

```
0   0   0
0   0   1
0   0   0
```

Observera, att i det här fallet har pixeln i sig ingen effekt på det slutgiltiga värdet för pixeln som visas på den platsen i bilden. Bara värdet för pixeln till höger används för att avgöra pixelns resultatvärde.

I ActionScript skapar du matrisen som en kombination av en Array-instans som innehåller värdena och två egenskaper som anger antalet rader och kolumner i matrisen. I följande exempel läses en bild in och när bilden är klar tillämpas ett faltningsfilter på bilden med matrisen i föregående exempel:

```
// Load an image onto the Stage.
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
{
    // Create the convolution matrix.
    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);
```

Det som inte visas i den här koden är effekten av att använda andra värden än 1 och 0 i matrisen. Samma matris med talet 8 i stället för 1 i högra positionen utför samma åtgärd (pixlarna flyttas till vänster). Dessutom påverkas bildens färger så att de blir åtta gånger intensivare. Det beror på att pixlarnas slutgiltiga färgvärden beräknas genom att matrisvärdena multipliceras med de ursprungliga pixelfärgerna. Värdena läggs ihop och divideras med värdet för filtrets `divisor`-egenskap. Observera att `divisor`-egenskapen är 1 i exempelkoden. I allmänhet bör du låta divisor-värdet vara lika med summan av matrisvärdena om du vill att färgernas intensitet ska vara ungefär densamma som i originalbilden. Om matrisens värden sammanlagt är 8 och divisor-egenskapen är 1 blir resultatbilden ungefär åtta gånger så intensiv som originalbilden.

Trots att matrisens effekt inte är särskilt märkbar kan andra matrisvärden användas för att skapa olika effekter. Här finns flera standarduppsättningar med matrisvärden för olika effekter med en 3 x 3-matris:

- Enkel oskärpa (divisor 5):

```
0 1 0
1 1 1
0 1 0
```

- Skärpa (divisor 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Kantidentifiering (divisor 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Reliefeffekt (divisor 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

Observera att divisor-egenskapen är 1 för de flesta av dessa effekter. Detta beror på att de negativa matrisvärdena plus de positiva matrisvärdena ger resultatet 1 (eller 0 i fallet kantidentifiering, men `divisor`-egenskapens värde kan inte vara 0).

Förskjutningsfilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen DisplacementMapFilter använder pixelvärden från ett BitmapData-objekt (kallas för förskjutningsbild) för att skapa en förskjutningseffekt på ett nytt objekt. Förskjutningsbilden är oftast annorlunda än det visningsobjekt eller den BitmapData-instans som filtret tillämpas på. En förskjutningseffekt innebär att pixlar förskjuts i den filtrerade bilden. M.a.o. flyttas de bort från den ursprungliga platsen i någon utsträckning. Filtret kan användas för att skapa en skiftad, förvrängd eller ojämn effekt.

Var och hur mycket förskjutning som används på en viss pixel avgörs av förskjutningsbildens färgvärde. När du arbetar med filtret anger du inte bara förskjutningsbild, du anger också följande värden som styr hur förskjutningen beräknas utifrån förskjutningsbilden:

- Mappningspunkt: Den plats i den filtrerade bilden där det övre vänstra hörnet av förskjutningsfiltret ska placeras. Använd värdet om du bara vill tillämpa filtret på en del av en bild.
- X-komponent: Vilken av förskjutningsbildens färgkanaler som ska påverka pixlarnas x-position.
- Y-komponent: Vilken av förskjutningsbildens färgkanaler som ska påverka pixlarnas y-position.
- X-skala: Ett multiplikatorvärde som anger hur stark förskjutningen är i x-axeln.
- Y-skala: Ett multiplikatorvärde som anger hur stark förskjutningen är i y-axeln.
- Filterläge: Avgör vad som ska ske i eventuella tomrum som kan uppstå när pixlar flyttas bort. Alternativen, som definieras som konstanter i klassen DisplacementMapFilterMode, är att visa de ursprungliga pixlarna (filterläget `IGNORE`), flytta över pixlarna till bildens andra sida (filterläget `WRAP`, som är standard), använda närmaste skiftade pixel (filterläget `CLAMP`) eller fylla i tomrummen med en färg (filterläget `COLOR`).

Ett enkelt exempel gör det lättare att förstå hur förskjutningsfilter fungerar. I koden läses en bild in. När den har lästs in helt centreras den på scenen och ett förskjutningsfilter tillämpas på den så att pixlarna i hela bilden flyttas vågrätt till vänster.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
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;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    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);
```

De egenskaper som används för att definiera förskjutningen är följande:

- Mappningsbild: Förskjutningsbitmappen är en ny BitmapData-instans som skapas av koden. Dess mått matchar den inlästa bildens mått (så att förskjutningen tillämpas på hela bilden). Den fylls i med heltäckande röda pixlar.
- Mappningspunkt: Värdet ställs in på punkten 0,0 så att förskjutningsfiltret täcker hela bilden.
- X-komponent: Det här värdet ställs in på konstanten `BitmapDataChannel.RED` så att mappningsbildens röda värde avgör hur mycket pixlarna förskjuts (hur långt de flyttas) på x-axeln.
- X-skala: Det här värdet är inställt på 250. Hela förskjutningen (från att förskjutningsbilden är helt röd) förskjuter bara bilden en liten bit (ungefär en halv pixel); om värdet är 1 flyttas bilden således endast 0,5 pixlar vågrätt. Värdet 250 gör att bilden flyttas ungefär 125 pixlar.

De här inställningarna gör att den filtrerade bildens pixlar flyttas 250 pixlar till vänster. Förflyttningens riktning (vänster eller höger) och mängden förflyttning baseras på färgvärdet i pixlarna i förskjutningsbilden. Filtret går igenom pixlarna i den filtrerade bilden en i taget (åtminstone pixlarna i det område som filtret tillämpas på, vilket i det här fallet är alla pixlar) och gör följande med varje pixel:

1. Söker efter motsvarande pixel i förskjutningsbilden. När filtret till exempel beräknar förskjutningsmängden för pixeln i det övre vänstra hörnet av den filtrerade bilden, läser det pixeln i övre vänstra hörnet på förskjutningsbilden.
2. Värdet avgörs för den angivna färgkanalen i förskjutningspixeln. I det här fallet är x-komponentens färgkanal den röda kanalen, så filtret kontrollerar vilket värde den röda kanalen i förskjutningsbilden har vid pixeln i fråga. Eftersom förskjutningsbilden är heltäckande röd är pixelns röda kanal 0xFF, d.v.s. 255. Det värdet används som förskjutningsvärde.
3. Jämför förskjutningsvärdet med mellanvärdet (127, d.v.s. mellan 0 och 255). Om förskjutningsvärdet är lägre än mellanvärdet flyttas pixeln i positiv riktning (till höger för x-förskjutning, nedåt för y-förskjutning). Om förskjutningsvärdet är högre än mellanvärdet (som är fallet i det här exemplet) flyttas pixeln i stället i negativ riktning (till vänster för x-förskjutning, uppåt för y-förskjutning). För att vara mer exakt subtraherar filtret förskjutningsvärdet från 127, och resultatet (positivt eller negativt) är den relativa förskjutning som tillämpas.
4. Slutligen avgörs den faktiska förskjutningsmängden genom beräkning av hur stor procentandel av fullständig förskjutning som det relativa förskjutningsvärdet representerar. I det här fallet är helt röd 100 % förskjutning. Den procentandelen multipliceras sedan med x-skalan eller y-skalan för att avgöra hur många pixlars förskjutning som ska tillämpas. I det här exemplet avgörs förskjutningsmängden av 100 % gånger multiplikatorn 250, vilket blir ungefär 125 pixlar till vänster.

Eftersom inga värden angavs för y-komponenten och y-skalan användes standardvärdena (vilket innebär att ingen förskjutning görs). Därför flyttas bilden inte alls i lodrät riktning.

Standardinställningen för filterläge, `WRAP`, används i det här exemplet så att de tomma pixlar som uppstår till höger fylls av de pixlar som flyttades utanför bildens vänsterkant. Du kan experimentera med det här värdet för att se de olika effekterna. Om du till exempel lägger till följande rad i den del av koden där förskjutningsegenskaperna ställs in (före raden `loader.filters = [displacementMap]`) ser bilden ut som om den har smetats ut på scenen:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

Följande kod är ett mer komplicerat exempel där ett förskjutningsfilter används för att skapa effekten av ett förstoringsglas på en bild:

```
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;

// Create the gradient circles that will together form the
// displacement map image
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);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
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();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    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;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

Koden skapar först två övertoningscirklar som kombineras till förskjutningsbilden. Den röda cirkeln skapar x-axelns förskjutning (`xyFilter.componentX = BitmapDataChannel.RED`) och den blå cirkeln skapar y-axelns förskjutning (`xyFilter.componentY = BitmapDataChannel.BLUE`). För att det ska bli lättare att förstå hur förskjutningsbilden ser ut läggs både originalcirklarna och den kombinerade cirkeln som fungerar som förskjutningsbild till längst ned på skärmen.

Sedan läses en bild in och när musen flyttas tillämpas förskjutningsfiltret på den del av bilden som är under muspekaren. Övertoningscirklarna som används som förskjutningsbild gör att det förskjutna området sprids ut från pekaren. Observera att de grå områdena i förskjutningsbilden inte orsakar någon förskjutning. Den grå färgen är `0x7F7F7F`. De blå och röda kanalerna för den nyansen av grått matchar mellannyansen för dessa färgkanaler exakt. Det gör att ingen förskjutning görs i grå områden i förskjutningsbilden. På samma sätt görs ingen förskjutning i cirkelns mitt. Färgen där är inte grå, men färgens blå kanal och röda kanal är identiska med den blå kanalen och den röda kanalen i mellangrå. Eftersom blå och röd är de färger som orsakar förskjutningen görs ingen förskjutning där.

Skuggningsfilter

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med klassen ShaderFilter kan du använda en anpassad filtereffekt som har definierats som en Pixel Bender-skuggning. Eftersom filtereffekten skrivs som en Pixel Bender-skuggning kan effekten anpassas fullständigt. Det filtrerade innehållet skickas till skuggningen som bildindata, och resultatet från skuggningsåtgärden blir filterresultatet.

Obs! *Skuggningsfiltret är tillgängligt i ActionScript från och med Flash Player 10 och Adobe AIR 1.5.*

Om du vill tillämpa ett skuggningsfilter på ett objekt måste du först skapa en Shader-instans som representerar Pixel Bender-skuggningen som du använder. Mer information om hur du skapar en Shader-instans och hur du anger indatabilds- och parametervärden finns i ”Arbeta med Pixel Bender-skuggningar” på sidan 301.

När du använder en skuggning som ett filter finns det tre viktiga saker som du måste ha i åtanke:

- Skuggningen måste ha definierats till att acceptera minst en indatabild.
- Det filtrerade objektet (visningsobjektet eller BitmapData-objektet som filtret tillämpas på) skickas till skuggningen som det första indatabildvärdet. Därför ska du inte manuellt ange något värde för första bildindata.

- Om skuggningen definierar fler än en indatabild, måste de övriga indata anges manuellt (d.v.s. egenskapen `input` ställs in för den ShaderInput-instans som tillhör Shader-instansen).

När du har ett Shader-objekt för skuggningen skapar du en ShaderFilter-instans. Detta är det faktiska filterobjektet som du använder på samma sätt som andra filter. Du skapar ett ShaderFilter som använder ett Shader-objekt genom att anropa `ShaderFilter()`-konstruktorn och skicka Shader-objektet som ett argument, vilket visas i detta exempel:

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Ett fullständigt exempel på hur du använder ett skuggningsfilter finns i ”Använda en skuggning som ett filter” på sidan 319.

Exempel på filtrering av visningsobjekt: Filter Workbench

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Filter Workbench finns ett användargränssnitt som du kan använda för att tillämpa olika filter på bilder och annat visuellt innehåll och se den kod som kan användas för att skapa samma effekt med ActionScript. Förutom ett verktyg för experimentering med filter, demonstrerar det här programmet följande tekniker:

- Skapa instanser av olika filter
- Tillämpa flera filter på ett visningsobjekt

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna för Filter Workbench finns i mappen Samples/FilterWorkbench. Programmet består av följande filer:

Fil	Beskrivning
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Den klass som innehåller programmets huvudsakliga funktionalitet, inklusive växling av vilket innehåll filter tillämpas på och tillämpning av filter på innehåll.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Ett gränssnitt som definierar vanliga metoder som implementeras av var och en av filterfabriksklasserna. I gränssnittet definieras de funktioner som används i klassen FilterWorkbenchController för att interagera med de enskilda filterfabriksklasserna.
i mappen com/example/programmingas3/filterWorkbench/: BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	En uppsättning klasser, som var och en implementerar gränssnittet IFilterFactory. Var och en av dessa klasser innehåller funktionaliteten för att skapa och ställa in värden för en viss typ av filter. På filteregenskapspanelerna i programmet används dessa fabriksklasser för att skapa instanser av motsvarande filter. Instanserna hämtas av klassen FilterWorkbenchController och tillämpas på bildinnehållet.

Fil	Beskrivning
com/example/programmingas3/filterWorkbench/IFilterPanel.as	Gränssnitt som definierar vanliga metoder för implementering av klasser som definierar användargränssnittspanelerna som används för att ändra filtervärden i programmet.
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	En verktygsklass som innehåller en metod som konverterar ett numeriskt färgvärde till ett hexadecimalt String-format
com/example/programmingas3/filterWorkbench/GradientColor.as	En klass som fungerar som ett värdeobjekt och kombinerar de tre värdena (color, alpha och ratio) som tillhör varje färg i GradientBevelFilter och GradientGlowFilter till ett enda objekt
Användargränssnitt (Flex)	
FilterWorkbench.mxml	Huvudfilen som definierar användargränssnittet för programmet.
flexapp/FilterWorkbench.as	En klass som innehåller funktionaliteten för huvudprogrammets användargränssnitt. Klassen används som klass med bakomliggande kod för MXML-programfilen.
I mappen flexapp/filterPanels: BevelPanel.mxml BlurPanel.mxml ColorMatrixPanel.mxml ConvolutionPanel.mxml DropShadowPanel.mxml GlowPanel.mxml GradientBevelPanel.mxml GradientGlowPanel.mxml	En uppsättning MXML-komponenter som innehåller funktionaliteten för varje panel som används för att ställa in alternativ för ett enda filter.
flexapp/ImageContainer.as	Ett visningsobjekt som fungerar som behållare för den inlästa bilden på skärmen
flexapp/controls/BGColorCellRenderer.as	Anpassad cellrenderare som används för att ändra bakgrundsfärg för en cell i komponenten DataGrid
flexapp/controls/QualityComboBox.as	Anpassad kontroll som definierar en kombinationsruta som kan användas för kvalitetsinställningen på olika filterpaneler.
flexapp/controls/TypeComboBox.as	Anpassad kontroll som definierar en kombinationsruta som kan användas för typinställningen på olika filterpaneler.
Användargränssnitt (Flash)	
FilterWorkbench.fla	Huvudfilen som definierar användargränssnittet för programmet.
flashapp/FilterWorkbench.as	En klass som innehåller funktionaliteten för huvudprogrammets användargränssnitt. Klassen används som dokumentklass för FLA-programfilen.

Fil	Beskrivning
I mappen flashapp/filterPanels: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	En uppsättning klasser som innehåller funktionaliteten för varje panel som används för att ställa in alternativ för ett enda filter. För varje klass finns också en associerad MovieClip-symbol i biblioteket i FLA-huvudfilen. Symbolens namn är samma som klassens (symbolen BlurPanel är till exempel kopplad till klassen som definieras i BlurPanel.as). Komponenterna som utgör användargränssnittet placeras och namnges i dessa symboler.
flashapp/ImageContainer.as	Ett visningsobjekt som fungerar som behållare för den inlästa bilden på skärmen
flashapp/BGColorCellRenderer.as	Anpassad cellrenderare som används för att ändra bakgrundsfärg för en cell i komponenten DataGrid
flashapp/ButtonCellRenderer.as	En anpassad cellrenderare som används för att inkludera en knappkomponent i en cell i komponenten DataGrid
Filtrerat bildinnehåll	
com/example/programmingas3/filterWorkbench/ImageType.as	Den här klassen fungerar som ett värdeobjekt som innehåller typ och URL för en enda bildfil som programmet kan läsa in och tillämpa filter på. Klassen innehåller också en uppsättning konstanter som representerar de faktiska bildfilerna.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Bilder och annat visuellt innehåll som filtren tillämpas på i programmet.

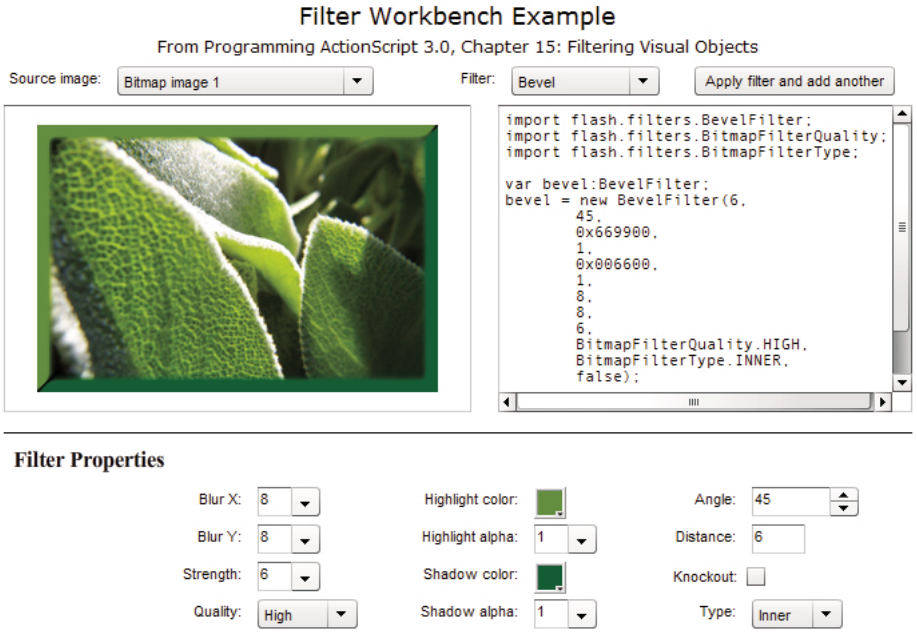
Experimentera med ActionScript-filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet Filter Workbench är utformat för att hjälpa dig att experimentera med olika filtereffekter och generera relevant ActionScript-kod för den effekten. Programmet gör att du kan välja på tre olika filer med visuellt innehåll, inklusive bitmappsbilder och en animering som har skapats i Flash, och tillämpa åtta olika ActionScript-filter på den valda bilden, antingen var för sig eller i kombination med andra filter. Programmet innehåller följande filter:

- Avfasning (flash.filters.BevelFilter)
- Oskärpa (flash.filters.BlurFilter)
- Färgmatris (flash.filters.ColorMatrixFilter)
- Faltning (flash.filters.ConvolutionFilter)
- Skugga (flash.filters.DropShadowFilter)
- Glöd (flash.filters.GlowFilter)
- Övertonad avfasning (flash.filters.GradientBevelFilter)
- Övertonad glöd (flash.filters.GradientGlowFilter)

När en användare har valt en bild och ett filter att tillämpa på den bilden visar programmet en panel med kontroller med vilka användaren kan ställa in det valda filtrets egenskaper. I följande bild visas programmet med avfasningsfiltret valt:



När användaren ändrar filteregenskaperna uppdateras förhandsvisningen i realtid. Användaren kan också tillämpa flera filter genom att anpassa ett filter, klicka på Apply filter and add another, anpassa ett filter till, klicka på Apply filter and add another igen, o.s.v.

Det finns några funktioner och begränsningar på programmets filterpaneler:

- Färgmatrisfiltret innehåller en uppsättning kontroller som direkt styr bildegenskaper som intensitet, kontraster, mättnad och nyans. Dessutom går det att ange anpassade färgmatrisvärden.
- Faltningsfiltret, som bara finns om ActionScript används, innehåller en uppsättning vanliga faltningsmatrisvärden. Det går också att ange egna värden. Till skillnad från klassen ConvolutionFilter, där en matris av valfri storlek kan användas, används en fast 3 x 3-matris i Filter Workbench eftersom det är den vanligaste filterstorleken.
- Förskjutningsfiltret och skuggningsfiltret, som bara finns i ActionScript, finns inte i Filter Workbench.

Skapa filterinstanser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Filter Workbench finns en uppsättning klasser, en för varje tillgängligt filter. Klasserna används på de olika panelerna för att skapa filtren. När en användare väljer ett filter skapas en instans av filterfabriksklassen av den ActionScript-kod som är kopplad till den filterpanelen. (Klasserna kallas *fabriksklasser* eftersom deras syfte är att skapa instanser av andra objekt, på ungefär samma sätt som en fabrik skapar enskilda produkter.)

Varje gång en användare ändrar ett egenskapsvärde på panelen anropas motsvarande metod i fabriksklassen av panelkoden. Varje fabriksklass innehåller specifika metoder som används av panelen för att skapa motsvarande filterinstans. Om användaren till exempel väljer oskärpefiltret skapas en BlurFactory-instans. I klassen BlurFactory finns metoden `modifyFilter()` som tar tre parametrar: `blurX`, `blurY` och `quality`. Tillsammans används de för att skapa BlurFilter-instansen:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Om användaren i stället väljer faltningsfiltret, får han eller hon ett filter med mycket större flexibilitet och därmed fler egenskaper att ställa in. I klassen ConvolutionFactory anropas följande kod när användaren väljer ett annat värde på filterpanelen:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                            matrixY:Number = 0,
                                            matrix:Array = null,
                                            divisor:Number = 1.0,
                                            bias:Number = 0.0,
                                            preserveAlpha:Boolean = true,
                                            clamp:Boolean = true,
                                            color:uint = 0,
                                            alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha,
clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Observera, att varje gång filtervärdena ändras, skickas en `Event.CHANGE`-händelse för att meddela avlyssnarna att filtrets värden har ändrats. Klassen FilterWorkbenchController, som är den klass som faktiskt tillämpar filtren på det filtrerade innehållet, avlyssnar den händelsen för att veta när den ska hämta en ny version av filtret och tillämpa den på det filtrerade innehållet.

Klassen FilterWorkbenchController behöver inte de specifika detaljerna i varje filterfabriksklass. Den behöver bara veta att filtret har ändrats och kunna komma åt en kopia av filtret. För att detta ska fungera innehåller programmet ett gränssnitt, IFilterFactory, som definierar vilket beteende en filterfabriksklass behöver ha för att programmets FilterWorkbenchController-instans ska kunna utföra sin uppgift. IFilterFactory definierar den `getFilter`()-metod som används i klassen FilterWorkbenchController:

```
function getFilter():BitmapFilter;
```

Observera att gränssnittsmetoden `getFilter()` definierar att den returnerar en BitmapFilter-instans och inte en viss typ av filter. Klassen BitmapFilter definierar inte en viss typ av filter. I stället är den en basklass som alla filterklasser bygger på. Varje filterfabriksklass definierar en viss implementering av metoden `getFilter()`. I den returneras en referens till filterobjektet som skapats av filterfabriksklassen. Här följer till exempel en förkortad version av källkoden för klassen ConvolutionFactory:

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

I klassen ConvolutionFactorys implementering av metoden `getFilter()` returneras en ConvolutionFilter-instans, med det registreras inte nödvändigtvis av alla objekt som anropar `getFilter()`. Enligt den definition av metoden `getFilter()` som ConvolutionFactory följer måste alla BitmapFilter-instanser returneras. Det kan vara en instans av vilken som helst av ActionScript-filterklasserna.

Tillämpa filter på visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som förklarats i tidigare används en instans av klassen FilterWorkbenchController (kallas hädanefter ”kontrollenhetsinstansen”) i programmet Filter Workbench. Det är kontrollenhetsinstansen som faktiskt tillämpar filter på det valda visningsobjektet. Innan kontrollenhetsinstansen kan tillämpa ett filter måste den ha information om vilken bild eller vilket visuellt innehåll filtret ska tillämpas på. När användaren väljer en bild anropas metoden `setFilterTarget()` i klassen FilterWorkbenchController och en av konstanterna som definierats i klassen ImageType skickas som parameter:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

Med den informationen läser kontrollenhetsinstansen in den angivna filen och sparar den i en instansvariabel med namnet `_currentTarget` när den har lästs in:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

När användaren väljer ett filter, anropar programmet kontrollenhetsinstansens `setFilter()`-metod och skickar en referens till det relevanta filterfabriksobjektet till kontrollenhetsinstansen. Referensen sparas sedan i en instansvariabel med namnet `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Observera, att enligt beskrivningen ovan vet inte kontrollenhetsinstansen vilken specifik datatyp filterfabriksinstansen har. Den vet bara att objektet implementerar IFilterFactory-instansen, vilket innebär att den har en `getFilter()`-metod och att händelsen `change` (`Event.CHANGE`) skickas när filtret ändras.

När användaren ändrar filtrets egenskaper på filterpanelen, registrerar kontrollenhetsinstansen att filtret har ändrats eftersom filterfabriksinstansens `change`-händelse anropar kontrollenhetsinstansens `filterChange()`-metod. Den metoden anropar i sin tur metoden `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

Arbetet med att tillämpa filtret på visningsobjektet görs i metoden `applyTemporaryFilter()`. Först får kontrollenheten en referens till filterobjektet genom att filterfabrikinstansens `getFilter()`-metod anropas.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

Kontrollenhetsinstansen har en Array-instansvariabel som heter `_currentFilters`. I den sparas alla filter som har tillämpats på visningsobjektet. Nästa steg är att lägga till det nyuppdaterade filtret i den arrayen:

```
_currentFilters.push(currentFilter);
```

Sedan tilldelar koden filterarrayen till visningsobjektets `filters`-egenskap som tillämpar filtren på bilden:

```
_currentTarget.filters = _currentFilters;
```

Slutligen, eftersom det senast tillagda filtret fortfarande är ”arbetsfiltret” ska det inte tillämpas permanent på visningsfiltret. Därför tas det bort från arrayen `_currentFilters`:

```
_currentFilters.pop();
```

Att filtret tas bort från arrayen påverkar inte det filtrerade visningsobjektet eftersom ett visningsobjekt skapar en kopia av filterarrayen när det tilldelas till egenskapen `filters`. I visningsobjektet används den interna arrayen i stället för den ursprungliga. Det innebär att ändringar som görs i en filterarray inte påverkar visningsobjektet förrän arrayen åter tilldelas visningsobjektets `filters`-egenskap.

Kapitel 15: Arbeta med Pixel Bender-skuggningar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med Adobes Pixel Bender-vektygslådan kan utvecklare skriva skuggningar som ger grafiska effekter och utför annan bild- och databearbetning. Pixel Bender-bytekoden kan köras i ActionScript om du vill tillämpa effekten på bilddata eller visuellt innehåll. Genom att använda Pixel Bender-skuggningar i ActionScript kan du skapa anpassade visuella effekter och utföra databearbetning som inte är möjlig med de inbyggda funktionerna i ActionScript.

Obs! *Pixel Bender stöds från och med Flash Player 10 och Adobe AIR 1.5. Pixel Bender-blandningar, -filter och -fyllningar stöds inte vid GPU-återgivning.*

Fler hjälpavsnitt

Adobe Pixel Bender Technology Center

Utvecklarhandbok för Pixel Bender

Pixel Bender-referens

flash.display.Shader

flash.filters.ShaderFilter

Pixel Bender-grunderna för Flash

Pixel Bender-grunderna för Flex

Grunderna i Pixel Bender-skuggningar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Adobe Pixel Bender är ett programmeringsspråk som används för att skapa eller manipulera bildinnehåll. Med Pixel Bender kan du skapa en kärna, som även kallas för en skuggning. Skuggningen definierar en enskild funktion som körs på varje pixel i en bild individuellt. Resultatet för varje anrop till funktionen är den utdatafärg som finns vid pixelkoordinaten i bilden. Du kan anpassa åtgärden genom att ange indatabilder och parametervärden. I en enda körning av en skuggning är inmatnings- och parametervärdena konstanta. Det enda som varierar är koordinaten för den pixel vars färg är resultatet av funktionsanropet.

När så är möjligt anropas skuggningsfunktionen för flera utdatapixelkoordinater parallellt. Då förbättras skuggningens prestanda och bearbetningen kan gå snabbare.

I ActionScript kan du enkelt skapa tre typer av effekter med hjälp av en skuggning:

- ritfyllning
- blandningsläge
- filter

En skuggning kan också köras i fristående läge. Om du använder fristående läge kan du komma åt skuggningens resultat direkt i stället för att behöva ange det tänkta användningsområdet i förväg. Resultatet kan nås som bilddata eller som binära data eller tasdata. Data behöver inte vara bilddata. På det här sättet kan du ge skuggningen en uppsättning data som indata. Skuggningen bearbetar data, och du får åtkomst till de resultatdata som returneras av skuggningen.

Pixel Bender stöds från och med Flash Player 10 och Adobe AIR 1.5. Pixel Bender-blandningar, -filter och -fyllningar stöds inte vid GPU-återgivning På mobilenheter fungerar Pixel Bender-skuggningar vid CPU-återgivning. Prestanda är dock inte lika goda som på stationära datorer. Många skuggningsprogram kanske bara körs med ett fåtal bildrutor per sekund.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du stöter på när du skapar och använder Pixel Bender-skuggningar:

Kärna För Pixel Bender är en kärna detsamma som en skuggning. Med Pixel Bender kan koden definiera en kärna, som definierar en enskild funktion som körs individuellt för varje pixel i en bild.

Pixel Bender-bytekod När en Pixel Bender-kärna kompileras omformas den till Pixel Bender-bytekod. Bytekoden nås och körs under körning.

Pixel Bender-språk Programmeringsspråket som används för att skapa en Pixel Bender-kärna.

Pixel Bender Toolkit Programmet som används för att skapa en Pixel Bender-bytekodfil från Pixel Bender-källkoden. Med verktygslådan kan du skriva, testa och kompilera Pixel Bender-källkoden.

Shader I det här dokumentet är en skuggning en uppsättning funktioner som skrivs med Pixel Bender-språket. En skuggnings kod är en visuell effekt eller något som utför en beräkning. I båda fallen returnerar skuggningen en uppsättning data (vanligtvis pixlarna i en bild). Skuggningen utför samma åtgärd på varje datapunkt. Den enda skillnaden är utdatapixelkoordinaterna. Skuggningen skrivs inte i ActionScript. Den skrivs i Pixel Bender-språket och kompileras till Pixel Bender-bytekod. Den kan bäddas in i en SWF-fil under kompileringen och läsas in som en extern fil under körning. I båda fallen nås den i ActionScript genom att ett Shader-objekt skapas och att det objektet länkas till skuggningens bytekod.

Skuggningsindata Komplexa indata, vanligtvis bitmappsbilddata, som används för beräkningar av en skuggning. För varje indatavariabel som definieras i en skuggning, används ett enskilt värde (det vill säga en enskild bild eller uppsättning binära data) för skuggningens hela körning.

Skuggningsparameter Ett enskilt värde (eller en begränsad uppsättning värden) som används i skuggningsberäkningar. Varje parametervärde definieras för en enskild skuggningskörning, och samma värde används under hela körningen.

Arbeta med kodexemplen

Det kan vara bra att testa kodexemplen som anges. Att testa koden innebär att köra den och gå igenom resultatet i SWF-filen som skapas. Alla exempel skapar innehåll med hjälp av drawing API som använder eller ändras av skuggningseffekten.

Den mesta exempelkoden omfattar två delar. Den ena delen är Pixel Bender-källkoden för skuggningen som används i exemplet. Du måste först använda Pixel Bender Toolkit för att kompilera källkoden till en Pixel Bender-bytekodfil. Skapa Pixel Bender-bytekodfilen genom att följa dessa steg:

1. Öppna Adobe Pixel Bender Toolkit. Välj om nödvändigt ”Turn on Flash Player warnings and errors” (”Aktivera Flash Player-indikatorer”) på menyn Build.
2. Kopiera Pixel Bender-kodexemplet och klistra in det i kodredigeringsfönstret i Pixel Bender Toolkit.

3 Välj ”Export kernel filter for Flash Player” (”Exportera kernelfilter för Flash Player”) på menyn File.

4 Spara Pixel Bender-bytekodfilen i samma katalog som Flash-dokumentet. Filens namn ska motsvara det namn som anges i exempelbeskrivningen.

ActionScript-delen i respektive exempel skrivs som en klassfil. Så här testar du exemplet i Flash Professional:

1 Skapa ett tomt Flash-dokument och spara det.

2 Skapa en ny ActionScript-fil och spara den i samma katalog som Flash-dokumentet. Filens namn måste överensstämma med namnet på klassen i koden. Om klassen MyApplication definieras i kodexemplet ska du använda namnet MyApplication.as för att spara ActionScript-filen.

3 Kopiera koden till ActionScript-filen och spara filen.

4 Klicka på en tom del av scenen eller arbetsytan i Flash-dokumentet för att aktivera dokumentets egenskapsinspektör.

5 Ange namnet på ActionScript-klassen som du kopierade från texten i fältet Dokumentklass i egenskapsinspektören.

6 Kör programmet med Kontroll > Testa filmen.

   Resultaten av exemplet visas i förhandsvisningsfönstret.

Dessa tekniker för att testa exempelkoder beskrivs närmare i ”Använda ActionScript-exempel” på sidan 1101.

Läsa in eller bädda in en skuggning

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Det första steget när du använder en Pixel Bender-skuggning i ActionScript är att få åtkomst till skuggningen i ActionScript-koden. Eftersom en skuggning skapas med Adobes Pixel Bender-verktygslådan, och skrivs i Pixel Bender-språket, kan det inte nås direkt i ActionScript. I stället skapar du en instans av klassen Shader som representerar Pixel Bender-skuggningen för ActionScript. Med Shader-objektet kan du ta reda på information om skuggningen, till exempel om den förväntar sig parametrar eller värden för indatabilder. Du skickar Shader-objektet till andra objekt när du vill använda skuggningen. Om du till exempel vill använda skuggningen som ett filter tilldelar du Shader-objektet till `shader`-egenskapen i ett ShaderFilter-objekt. Om du i stället vill använda skuggningen som en fyllning skickar du Shader-objektet som ett argument till metoden `Graphics.beginShaderFill()`.

ActionScript-koden får tillgång till en skuggning som skapats med Adobes Pixel Bender-verktygslådan (en .pbj-fil) på två sätt:

- Inläsning vid körning: skuggningsfilen kan läsas in som en extern resurs med hjälp av ett URLLoader-objekt. Den här metoden påminner om att läsa in en extern resurs, till exempel en textfil. Följande exempel visar hur en skuggningsbytekodfil läses in vid körning och länkas till en Shader-instans:

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- Inbäddad i SWF-filen: skuggningsfilen kan bäddas in i SWF-filen vid kompilering med hjälp av metadatataggen `[Embed]`. Metadatataggen `[Embed]` är bara tillgänglig om du använder Flex SDK för att kompilera SWF-filen. `[Embed]`-taggens `source`-parameter pekar på skuggningsfilen och dess `mimeType`-parameter är `"application/octet-stream"`, vilket visas i det här exemplet:

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shader:Shader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

I båda fallen länkar du bytekoden för rådataskuggning (`URLLoader.data`-egenskapen eller en instans av `[Embed]`-dataklassen) till Shader-instansen. Så som visas i exemplen ovan kan du tilldela bytekoden till Shader-instansen på två sätt. Du kan skicka skuggningsbytekoden som ett argument till `Shader()`-konstruktorn. Du kan också ange den som `byteCode`-egenskap för Shader-instansen.

När en Pixel Bender-skuggning har skapats och länkats till ett Shader-objekt, kan du använda skuggningen för att skapa effekter på olika sätt. Du kan använda den som ett filter, ett blandningsläge, en bitmappsfyllning eller för fristående bearbetning av bitmapps- eller andra data. Du kan också använda Shader-objektets `data`-egenskap när du vill komma åt skuggningens metadata, ange indatabilder och ställa in parametervärden.

Få tillgång till skuggningens metadata

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När utvecklaren skapar en Pixel Bender-skuggningskärna, kan han eller hon ange metadata om skuggningen i Pixel Bender-källkoden. När du använder en skuggning i ActionScript kan du undersöka skuggningen och hämta dess metadata.

När du skapar en Shader-instans och länkar den till en Pixel Bender-skuggning, skapas ett ShaderData-objekt som innehåller data om skuggningen och lagras i Shader-objektets `data`-egenskap. Klassen ShaderData definierar inte några egenskaper på egen hand. Under körning läggs en egenskap till dynamiskt i ShaderData-objektet för varje metadatavärde som definieras i skuggningens källkod. Namnet som ges till respektive egenskap är detsamma som namnet som anges i metadata. Anta att källkoden för en Pixel Bender-skuggning omfattar följande metadatadefinition:

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

ShaderData-objektet skapas för den skuggningen med följande egenskaper och värden:

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"`
- `version` (String): `"1"`
- `description` (String): `"Creates a version of the specified image with the specified brightness"`

Eftersom metadataegenskaperna läggs till dynamiskt i ShaderData-objektet kan du använda en `for..in`-slinga när du undersöker ShaderData-objektet. Med den här tekniken kan du ta reda på om skuggningen har några metadata och vilka metadatavärdena är. Förutom metadataegenskaper kan ett ShaderData-objekt ha egenskaper som representerar indata och parametrar som definieras i skuggningen. När du använder en `for..in`-slinga för att undersöka ett ShaderData-objekt kontrollera du datatypen för varje egenskap för att undersöka om egenskapen är indata (en ShaderInput-instans), en parameter (en ShaderParameter-instans) eller ett metadatavärde (en String-instans). Följande exempel visar hur du använder en `for..in`-slinga för att undersöka de dynamiska egenskaperna i en skuggnings `data`-egenskap. Varje metadatavärde läggs till i en Vector-instans med namnet `metadata`. I det här exemplet förutsätts det att en Shader-instans med namnet `myShader` redan har skapats:

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

En version av det här exemplet som också extraherar skuggningsindata och parametrar finns i ”Identifiera skuggningsindata och -parametrar” på sidan 306. Mer information om indata och parameteregenskaper finns i ”Ange skuggningsindata och parametervärden” på sidan 306.

Ange skuggningsindata och parametervärden

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Många Pixel Bender-skuggningar definieras till att använda en eller flera indatabilder som används i skuggningsbearbetningen. Det är till exempel vanligt att en skuggning accepterar en källbild och skapar utdata av den bilden med en viss effekt. Beroende på hur skuggningen används kan indatavärdet anges automatiskt eller så måste du explicit ange ett värde. Många skuggningar anger på ett liknande sätt parametrar som används för att anpassa skuggningens utdata. Du måste också explicit ange ett värde för varje parameter innan du använder skuggningen.

Du använder Shader-objektets `data`-egenskap för att ange skuggningsindata och -parametrar och för att ta reda på om en viss skuggning förväntar sig indata eller parametrar. Egenskapen `data` är en ShaderData-instans.

Identifiera skuggningsindata och -parametrar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Det första steget i att ange skuggningsindata och -parametervärden är att ta reda på om skuggningen som du använder förväntar sig några indatabilder eller parametrar. Varje Shader-instans har en `data`-egenskap som innehåller ett ShaderData-objekt. Om skuggningen definierar några indata eller parametrar, nås de som egenskaper i det ShaderData-objektet. Egenskapens namn matchar namnen som anges för indata och parametrar i skuggningens källkod. Om en skuggning till exempel definierar indata med namnet `src` har ShaderData-objektet en egenskap med namnet `src` som representerar dessa indata. Varje egenskap som representerar indata är en ShaderInput-instans, och varje egenskap som representerar en parameter är en ShaderParameter-instans.

Det bästa är att den som utvecklar skuggningen dokumenterar den, där det anges vilka indatavärden och parametrar som skuggningen förväntar sig, vad de representerar, lämpliga värden o.s.v.

Om skuggningen emellertid inte dokumenteras (och du inte har källkoden) kan du granska skuggningens data för att på så sätt identifiera indata och parametrar. Egenskaperna som representerar indata och parametrar läggs till dynamiskt i ShaderData-objektet. Du kan därför använda en `for..in`-slinga när du undersöker ShaderData-objektet för att ta reda på om dess tillhörande skuggning definierar några indata eller parametrar. Så som beskrivs i ”Få tillgång till skuggningens metadata” på sidan 304 kan alla metadatavärden som definieras för en skuggning också nås som en dynamisk egenskap som läggs till i egenskapen `Shader.data`. När du använder den här metoden för att identifiera skuggningens indata och parametrar, kontrollerar du datatypen för de dynamiska egenskaperna. Om en egenskap är en ShaderInput-instans representerar den indata. Om den är en ShaderParameter-instans representerar den en parameter. Annars är den ett metadatavärde. Följande exempel visar hur du använder en `for..in`-slinga för att undersöka de dynamiska egenskaperna i en skuggnings `data`-egenskap. Varje indata (ShaderInput-objekt) läggs till i en Vector-instans med namnet `inputs`. Varje parameter (ShaderParameter-objekt) läggs till i en Vector-instans med namnet `parameters`. Alla metadataegenskaper läggs slutligen till i en Vector-instans med namnet `metadata`. I det här exemplet förutsätts det att en Shader-instans med namnet `myShader` redan har skapats:

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

Ange indatavärden för skuggningen

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Många skuggningar förväntar en eller flera indatabilder som ska användas i skuggningsbearbetningen. I många fall anges indata automatiskt när Shader-objektet används. Anta att en skuggning kräver ett indatavärde, och att skuggningen används som ett filter. När filtret används på ett visningsobjekt eller BitmapData-objekt, ställs det objektet in automatiskt som indata. I så fall behöver du inte ställa in indatavärdet explicit.

I vissa fall, speciellt om en skuggning definierar flera indatavärden, måste du emellertid ställa in ett värde explicit. Varje indatavärde som definieras i en skuggning representeras i ActionScript med ett ShaderInput-objekt. ShaderInput-objektet är en egenskap i ShaderData-instansen i Shader-objektets `data`-egenskap, vilket beskrivs i ”Identifiera skuggningsindata och -parametrar” på sidan 306. Anta att en skuggning definierar ett indatavärde med namnet `src`, och att skuggningen länkas till ett Shader-objekt med namnet `myShader`. I så fall får du tillgång till det ShaderInput-objekt som motsvarar `src`-indatavärdet med följande identifierare:

```
myShader.data.src
```

Varje ShaderInput-objekt har en `input`-egenskap som används för att ställa in värdet på indata. Du ställer in egenskapen `input` till en BitmapData-instans när du vill ange bilddata. Du kan också ställa in `input`-egenskapen till en BitmapData eller Vector.<Number>-instans om du vill ange binära eller numeriska data. Mer information och restriktioner för hur du använder en BitmapData- eller Vector.<Number>-instans som indata. Läs mer i avsnittet om `ShaderInput.input` i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Förutom `input`-egenskapen har ett ShaderInput-objekt egenskaper som kan användas för att ta reda på vilken typ av bild som indata förväntar sig. Dessa egenskaper är bland andra `width`, `height` och `channels`. Varje ShaderInput-objekt har också en `index`-egenskap, som är användbar när du vill ta reda på om ett explicit värde måste anges för indata. Om en skuggning förväntar fler indata än det antal som ställs in automatiskt, ställer du in värden för dessa indata. Mer information om hur du kan använda en skuggning på olika sätt, och om indatavärdena ställs in automatiskt, finns i ”Använda en skuggning” på sidan 311.

Ange parametervärden för skuggningen

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Vissa skuggningar definierar parametervärden som skuggningen använder för att skapa resultatet. En skuggning som till exempel ändrar intensiteten för en bild kanske anger en intensitetsparameter som bestämmer hur mycket åtgärden påverkar intensiteten. En enda parameter som definieras i en skuggning kan förvänta ett enda värde eller flera värden, enligt parameterdefinition i skuggningen. Varje parameter som definieras i en skuggning representeras i ActionScript med ett ShaderParameter-objekt. ShaderParameter-objektet är en egenskap till ShaderData-instansen i Shader-objektets dataegenskap, vilket beskrivs i ”Identifiera skuggningsindata och -parametrar” på sidan 306. Anta att en skuggning definierar en parameter med namnet `brightness`, och att skuggningen representeras av ett Shader-objekt med namnet `myShader`. I så fall får du tillgång till den ShaderParameter som motsvarar `brightness`-parametern med följande identifierare:

```
myShader.data.brightness
```

Om du vill ställa in ett eller flera värden för parametern skapar du en ActionScript-array som innehåller värdet eller värdena och tilldelar den till ShaderParameter-objektets `value`-egenskap. Egenskapen `value` definieras som en Array-instans eftersom det är möjligt att en enda skuggningsparameter kräver flera värden. Även om skuggningsparametern bara förväntar sig ett enda värde, måste du placera värdet i ett Array-objekt för att kunna tilldela den till egenskapen `ShaderParameter.value`. I exemplet nedan visas hur du ställer in ett enda värde som egenskapen `value`:

```
myShader.data.brightness.value = [75];
```

Om Pixel Bender-källkoden för skuggningen definierar ett standardvärde för parametern, skapas en array med standardvärdet eller -värdena och den tilldelas till ShaderParameter-objektets `value`-egenskap när Shader-objektet skapas. När en array har tilldelats till `value`-egenskapen (inklusive om det är standardarrayen) kan parametervärdet ändras genom att värdet på arrayelementet ändras. Du behöver inte skapa en ny array och tilldela den till `value`-egenskapen.

I exemplet nedan visas hur du ställer in en skuggnings parametervärde i ActionScript. I det här exemplet definierar skuggningen en parameter med namnet `color`. Parametern `color` deklareras som en `float4`-variabel i Pixel Bender-källkoden, vilket innebär att det är en array med fyra flyttal. I exemplet ändras `color`-parametervärdet fortlöpande, och varje gång det ändras används skuggningen för att rita en färgad rektangel på skärmen. Resultatet blir en animerad färgändring.

Obs! *Koden i det här exemplet har skrivits av Ryan Taylor. Vi tackar Ryan för att han delat med sig. Fler exempel på Ryans arbete och kod finns på www.boostworthy.com/.*

ActionScript-koden centreras kring tre metoder:

- `init()`: I metoden `init()` läser koden in Pixel Bender-bytekodfilen som innehåller skuggningen. När filen läses in anropas metoden `onLoadComplete()`.
- `onLoadComplete()`: I metoden `onLoadComplete()` skapar koden Shader-objektet med namnet `shader`. Den skapar också en Sprite-instans med namnet `texture`. I metoden `renderShader()` ritar koden skuggningsresultatet i `texture` en gång per bildruta.
- `onEnterFrame()`: Metoden `onEnterFrame()` anropas en gång per bildruta, vilket skapar animeringseffekten. I den här metoden ställer koden in skuggningsparameterns värde till den nya färgen och anropar sedan metoden `renderShader()` för att rita skuggningsresultatet som en rektangel.
- `renderShader()`: I metoden `renderShader()` anropar koden metoden `Graphics.beginShaderFill()` för att ange en skuggningsfyllning. Den ritar sedan en rektangel vars fyllning definieras av skuggningsresultatet (den färg som skapas). Mer information om hur du använder en skuggning på det här sättet finns i ”Använda en skuggning som teckningsfyllning” på sidan 311.

Nedan finns ActionScript-koden för det här exemplet. Använd den här klassen som huvudprogramklass för ett ActionScript-projekt i Flash Builder, eller som dokumentklass för FLA-filen i Flash Professional:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
```

```
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

Nedan finns källkoden för ColorFilter-skuggningskärnan, som används för att skapa Pixel Bender-filen med bytekod, ”ColorFilter.pbj”:

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

Om du använder en skuggning vars parametrar inte dokumenteras kan du ta reda på hur många element och av vilken typ som måste inkluderas i arrayen genom att kontrollera ShaderParameter-objektets `type`-egenskap. Egenskapen `type` anger parameterns datatyp, som definieras i själva skuggningen. En lista med antal och typ av element som förväntas av respektive parametertyp finns i egenskapslistan för `ShaderParameter.value` i referensinformationen till ActionScript 3.0.

Varje ShaderParameter-objekt har också en `index`-egenskap, som anger var parametern passar in i ordningen för skuggningens parametrar. Förutom dessa egenskaper kan ett ShaderParameter-objekt ha fler egenskaper som innehåller metadatavärden som anges av den som utvecklat skuggningen. Utvecklaren kan till exempel ange metadatavärden som minsta, högsta och standardvärden för en parameter. Alla metadatavärden som utvecklaren anger läggs till i ShaderParameter-objektet som dynamiska egenskaper. Du undersöker dessa egenskaper genom att göra en slinga över ShaderParameter-objektets dynamiska egenskaper med en `for..in`-slinga för att identifiera dess metadata. Följande exempel visar hur du använder en `for..in`-slinga för att identifiera ett ShaderParameter-objekts metadata. Varje metadatavärde läggs till i en Vector-instans med namnet `metadata`. Observera att det här exemplet förutsätter att en Shader-instans med namnet `myShader` redan har skapats, och att det är känt att den har en parameter med namnet `brightness`:

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

Använda en skuggning

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När en Pixel Bender-skuggning är tillgänglig i ActionScript som ett Shader-objekt, kan den används på flera sätt:

- Skuggning av teckningsfyllningar: Skuggningen definierar fyllningsdelen i en form som har ritats med teckningsgränssnittet
- Blandningsläge: Skuggningen definierar blandningen mellan två överlappande visningsobjekt
- Filter: Skuggningen definierar ett filter som ändrar utseendet på det visuella innehållet
- Fristående skuggningsbearbetning: Skuggningsbearbetningen körs utan att det tänkta användningsområdet för resultatet anges. Skuggningen kan också köras i bakgrunden, då resultatet blir tillgängligt när bearbetningen är klar. Den här metoden kan användas för att skapa bitmappsdata och även för att bearbeta ickevisuella data.

Använda en skuggning som teckningsfyllning

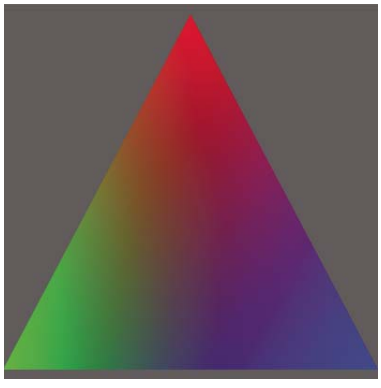
Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du lägger till en skuggning för att skapa en ritfyllning, skapar du en vektorform med drawing API-metoderna (teckningsgränssnittet). Skuggningens utdata används för att fylla formen, på samma sätt som en bitmappsbild kan användas som bitmappsfyllning med drawing API (teckningsgränssnittet). Om du vill skapa en skuggningsfyllning anropar du Graphics-objektets `beginShaderFill()`-metod vid den punkt i koden där du vill börja rita formen. Skicka Shader-objektet som det första argumentet till metoden `beginShaderFill()`, så som visas i exemplet:

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

När du använder en skuggning som ritfyllning, ska du ställa in alla indatabildvärden och parametervärden som krävs för skuggningen.

I exemplet nedan visas hur du använder en skuggning som ritfyllning. I det här exemplet skapar skuggningen en övertoning med tre punkter. Övertoningen har tre färger, en vid vart och ett av triangelhörnen, med en övertoningsblandning mellan dem. Färgerna roterar dessutom för att skapa en animerad snurrande färgeffekt.



Obs! *Koden i det här exemplet har skrivits av Petri Leskinen. Vi tackar Petri för att han delat med sig. Fler exempel på Petris arbeten och självstudiekurser finns på http://pixelero.wordpress.com/.*

ActionScript-koden finns i tre metoder:

- `init()`: Metoden `init()` anropas när programmet läses in. I den här metoden anger koden startvärdena för Point-objekten som representerar hörnen i triangeln. Koden skapar också en Sprite-instans med namnet `canvas`. I metoden `updateShaderFill()` ritar koden sedan skuggningsresultatet i `canvas` en gång per bildruta. Koden läser slutligen in skuggningens bytekodfil.
- `onLoadComplete()`: I metoden `onLoadComplete()` skapar koden Shader-objektet med namnet `shader`. Den ställer också in startparametervärden. Koden lägger slutligen till metoden `updateShaderFill()` som avlyssnare för händelsen `enterFrame`, vilket innebär att den anropas en gång per bildruta för att skapa en animeringseffekt.
- `updateShaderFill()`: Metoden `updateShaderFill()` anropas en gång per bildruta, vilket skapar animeringseffekten. I den här metoden beräknar koden skuggningsparameterns värden och ställer in dem. Koden anropar sedan metoden `beginShaderFill()` för att skapa en skuggningsfyllning och anropar andra ritnings-API-metoder för att rita skuggningsresultatet i en triangel.

Nedan finns ActionScript-koden för det här exemplet. Använd den här klassen som huvudprogramklass för ett ActionScript-projekt i Flash Builder, eller som dokumentklass för en FLA-fil i Flash Professional:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle.y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener(Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
```

```
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120);

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

Nedan finns källkoden för ThreePointGradient-skuggningskärnan, som används för att skapa Pixel Bender-bytekodfilen, ”ThreePointGradient.pbj”:

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
```

```
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;

    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

***Obs!** Om du använder en skuggfyllning vid GPU-återgivning blir det fyllda området cyanfärgat.*

Mer information om hur du ritar skuggningar med hjälp av drawing API (teckningsgränssnittet) finns i ”Använda Drawing API (teckningsgränssnittet)” på sidan 222.

Använda en skuggning som blandningsläge

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du använder en skuggning som blandningsläge på samma sätt som andra blandningslägen. Skuggningen definierar utseendet som bildas från två visningsobjekt som blandas ihop visuellt. Om du vill använda en skuggning som blandningsläge tilldelar du Shader-objektet till egenskapen `blendShader` för visningsobjektet i förgrunden. Om du tilldelar ett annat värde än `null` till egenskapen `blendShader` anges visningsobjektets `blendMode`-egenskap automatiskt till `BlendMode.SHADER`. I exemplet nedan visas hur du använder en skuggning som blandningsläge. Observera att det här exemplet förutsätter att det finns ett visningsobjekt med namnet `foreground` i samma överordnade objekt i visningslistan som annat visningsinnehåll, där `foreground` överlappar annat innehåll:

```
foreground.blendShader = myShader;
```

När du använder en skuggning som blandningsläge måste skuggningen definieras med minst två punkter. Så som exemplet visar ska du inte ställa in indatavärdena i koden. I stället används de två blandningsbilderna automatiskt som skuggningsindata. Förgrundsbilden ställs in som den andra bilden. (Detta är visningsobjektet som blandningsläget används på.) En bakgrundsbild skapas genom att alla pixlar bakom förgrundsbildens begränsningsram sätts samman. Den här bakgrundsbilden ställs in som den första indatabilden. Om du använder en skuggning som förväntar fler än två indata, anger du ett värde för alla indata utöver de första två.

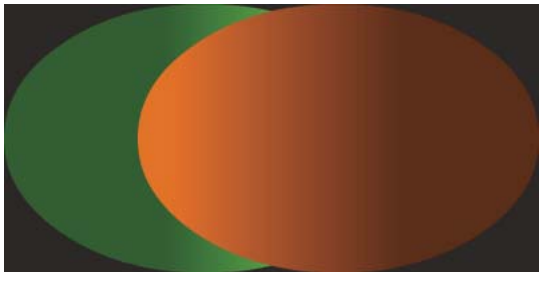
I exemplet nedan visas hur du använder en skuggning som blandningsläge. I det här exemplet används ett ljusare blandningsläge som baseras på luminiscens. Resultatet från blandningen blir att det ljusaste pixelvärdet från något av de blandade objekten blir pixeln som visas.

Obs! *Koden i det här exemplet har skrivits av Mario Klingemann. Vi tackar Mario för att han delat med sig. Mer av Marios arbete finns på www.quasimondo.com/.*

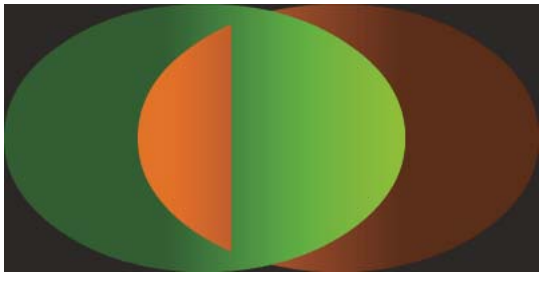
Den viktiga ActionScript-koden finns i dessa två metoder:

- `init()`: Metoden `init()` anropas när programmet läses in. I den här metoden läser koden in skuggningens bytekodfil.
- `onLoadComplete()`: I metoden `onLoadComplete()` skapar koden Shader-objektet med namnet `shader`. Sedan ritas tre objekt. Det första, `backdrop`, är en mörkgrå bakgrund bakom de blandade objekten. Det andra, `backgroundShape`, är en grön övertoningsellips. Det tredje objektet, `foregroundShape`, är en orange övertoningsellips.

  `foregroundShape`-ellipsen är blandningens förgrundsobjekt. Blandningens bakgrundsbild utgörs av den del av `backdrop` och den del av `backgroundShape` som överlappas av `foregroundShape`-objektets begränsningsram. Objektet `foregroundShape` är det objekt som finns längst fram i visningslistan. Det överlappar delvis `backgroundShape` och överlappar helt `backdrop`. På grund av den här överlappningen visas den orange ellipsen (`foregroundShape`) helt och hållet och en del av den gröna ellipsen (`backgroundShape`) döljs av den, om inget blandningsläge används:



  När blandningsläget används lyser den ljusare delen av den gröna ellipsen emellertid igenom, eftersom den är ljusare än den del av `foregroundShape` som överlappar den:



Nedan finns ActionScript-koden för det här exemplet. Använd den här klassen som huvudprogramklass för ett ActionScript-projekt i Flash Builder, eller som dokumentklass för FLA-filen i Flash Professional:

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
```

```
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

Nedan finns källkoden för LumaLighten-skuggningskärnan, som används för att skapa Pixel Bender-filen med bytekod, ”LumaLighten.pbj”:

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

Mer information om blandningslägen finns i ”Använda blandningslägen” på sidan 186.

Obs! *När du kör ett Pixel Bender-skuggningsprogram som en blandning i Flash Player eller AIR beter funktionerna sampling och* `outCoord()` *sig annorlunda än i andra sammanhang. I en blandning returnerar en samplingsfunktion alltid den pixel som utvärderas av skuggningen. Du kan till exempel inte använda en förskjutning av* `outCoord()` *för att sampla en intilliggande pixel. Och om du använder funktionen* `outCoord()` *utanför en samplingsfunktion utvärderas dess koordinater alltid till 0. Du kan till exempel inte använda en pixels position för att påverka hur de blandade bilderna kombineras.*

Använda en skuggning som ett filter

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Att använda en skuggning som ett filter går till ungefär som att använda andra filter i ActionScript. När du använder en skuggning som ett filter skickas den filtrerade bilden (ett visningsobjekt eller BitmapData-objekt) till skuggningen. Skuggningen använder indatabilden för att skapa filterutdata, som vanligtvis är en modifierad version av den ursprungliga bilden. Om det filtrerade objektet är ett visningsobjekt visas skuggningens utdata på skärmen i stället för det filtrerade visningsobjektet. Om det filtrerade objektet är ett BitmapData-objekt blir skuggningens utdata innehållet i det BitmapData-objekt vars `applyFilter()`-metod anropas.

Om du vill använda en skuggning som filter måste du först skapa Shader-objektet på det sätt som beskrivs i ”Läsa in eller bädda in en skuggning” på sidan 303. Sedan skapar du ett ShaderFilter-objekt som länkas till Shader-objektet. ShaderFilter-objektet är filtret som används på det filtrerade objektet. Du tillämpar det på ett objekt på samma sätt som du tillämpar andra filter. Du skickar det till egenskapen `filters` för ett visningsobjekt eller anropar metoden `applyFilter()` för ett BitmapData-objekt. Följande exempelkod skapar ett ShaderFilter-objekt och använder filtret på visningsobjektet med namnet `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

När du använder en skuggning som filter måste skuggningen definieras med minst en inmatning. Så som exemplet visar ska du inte ställa in indatavärdet i koden. I stället ställs det filtrerade visningsobjektet eller BitmapData-objektet in som indatabilden. Om du använder en skuggning som förväntar fler än en inmatning, anger du ett värde för alla inmatningar utöver den första.

I vissa fall ändrar ett filter dimensionerna för den ursprungliga bilden. En vanlig skuggeffekt lägger till exempel till extra pixlar som innehåller skuggan som läggs till i bilden. När du använder en skuggning som ändrar bildens dimensioner ställer du in egenskaperna `leftExtension`, `rightExtension`, `topExtension` och `bottomExtension` för att ange hur mycket bilden ska ändra storlek.

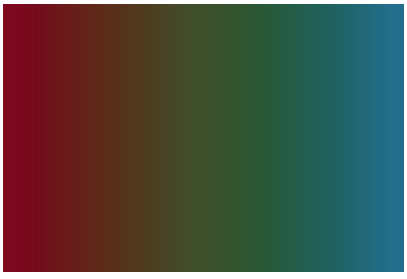
I exemplet nedan visas hur du använder en skuggning som filter. Filtret i det här exemplet inverterar de röda, gröna och blå kanalvärdena för en bild. Resultatet blir en ”negativ” version av bilden.

Obs! *Skuggningen som det här exemplet använder är Pixel Bender-kärnan invertRGB.pbk som ingår i Pixel Bender Toolkit. Du kan läsa in källkoden för kärnan från installationskatalogen för Pixel Bender Toolkit. Kompilera källkoden och spara bytekodfilen i samma katalog som källkoden.*

Den viktiga ActionScript-koden finns i dessa två metoder:

- `init()`: Metoden `init()` anropas när programmet läses in. I den här metoden läser koden in skuggningens bytekodfil.

- `onLoadComplete()`: I metoden `onLoadComplete()` skapar koden Shader-objektet med namnet `shader`. Sedan skapas och ritas innehållet i ett objekt med namnet `target`. Objektet `target` är en rektangel som fylls med en linjär övertoningsfärg som är röd till vänster, gulgrön i mitten och ljusblå till höger. Det ofiltrerade objektet ser ut så här:



  När filtret som används på färgen inverteras ser rektangeln ut så här:

Skuggningen som det här exemplet använder är Pixel Bender-kärnan invertRGB.pbk som ingår i Pixel Bender Toolkit. Källkoden finns i filen invertRGB.pbk i installationskatalogen för Pixel Bender Toolkit. Kompilera källkoden och spara bytekodfilen med namnet invertRGB.pbj som finns i samma katalog som ActionScript-källkoden.

Nedan finns ActionScript-koden för det här exemplet. Använd den här klassen som huvudprogramklass för ett ActionScript-projekt i Flash Builder, eller som dokumentklass för FLA-filen i Flash Professional:

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
```

```
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

Mer information om att använda filter finns i ”Skapa och tillämpa filter” på sidan 269.

Använda en skuggning i fristående läge

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du använder en skuggning i fristående läge körs skuggningsbearbetningen oberoende av hur du tänker använda resultatet. Du anger vilken skuggning som ska köras, ställer in indata- och parametervärden och anger vilket objekt som resultatdata ska placeras i. Du kan använda en skuggning i fristående läge av två anledningar:

- Bearbeta andra data än bilddata: I fristående läge kan du välja att skicka godtyckliga binära data eller taldata till skuggningen i stället för bitmappsbilddata. Du kan välja att skuggningsresultatet returneras som binära data eller taldata förutom som bitmappsdata.
- Bakgrundsbearbetning: När du kör en skuggning i fristående läge körs skuggningen som standard asynkront. Det innebär att skuggningen körs i bakgrunden medan programmet fortsätter att köras, och koden meddelas när skuggningsbearbetningen är klar. Du kan använda en skuggning som tar längre tid att köra och inte låser programmets användargränssnitt eller annan bearbetning medan skuggningen körs.

Med ett ShaderJob-objekt kan du köra en skuggning i fristående läge. Först skapar du ShaderJob-objektet och länkar det till Shader-objektet som representerar skuggningen som ska köras:

```
var job:ShaderJob = new ShaderJob(myShader);
```

Sedan ställer du in alla indata- eller parametervärden som skuggningen förväntar. Om du kör skuggningen i bakgrunden registrerar du även en avlyssnare för ShaderJob-objektets `complete`-händelse. Avlyssnaren anropas när skuggningen avslutas:

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

Sedan skapar du ett objekt där skuggningsåtgärdens resultat sparas när åtgärden är klar. Du tilldelar det objektet till ShaderJob-objektets `target`-egenskap:

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Tilldela en BitmapData-instans till `target`-egenskapen om du använder ShaderJob för att bearbeta bilder. Om du bearbetar binära data eller taldata tilldelar du ett ByteArray-objekt eller en Vector.<Number>-instans till egenskapen `target`. I så fall måste du ställa in ShaderJob-objektets `width`- och `height`-egenskaper så att de anger den mängd data som ska skickas till `target`-objektet.

Obs! *Du kan ställa in ShaderJob-objektets `shader`-, `target`-, `width`- och `height`-egenskaper i ett enda steg genom att skicka argumenten till `ShaderJob()`-konstruktorn, så här:`var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`*

När du är klar att köra skuggningen anropar du ShaderJob-objektets `start()`-metod:

```
job.start();
```

Om du anropar `start()` körs ShaderJob som standard asynkront. I så fall fortsätter programkörningen omedelbart med nästa kodrad i stället för att vänta på att skuggningen ska bli klar. När skuggningsåtgärden är klar anropar ShaderJob-objektet sina `complete`-händelseavlyssnare och meddelar dem att åtgärden är klar. Vid den här punkten (det vill säga i brödtexten till `complete`-händelseavlyssnaren) innehåller `target`-objektet skuggningsåtgärdens resultat.

Obs! *I stället för att använda `target`-egenskapsobjektet kan du hämta skuggningsresultatet direkt från det händelseobjekt som skickas till avlyssnarmetoden. Händelseobjektet är en ShaderEvent-instans. ShaderEvent-objektet har tre egenskaper som du kan använda för att komma åt resultatet, beroende på datatypen för det objekt du anger som `target`-egenskap: `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` och `ShaderEvent.vector`.*

Du kan också skicka ett `true`-argument till metoden `start()`. I så fall körs skuggningsåtgärden synkront. All kod (inklusive interaktion med användargränssnittet och andra händelser) gör en paus medan skuggningen körs. När skuggningen är klar innehåller `target`-objektet skuggningsresultatet och programmet fortsätter med nästa kodrad.

```
job.start(true);
```

Kapitel 16: Arbeta med filmklipp

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen MovieClip är huvudklass för animerings- och filmklippssymboler som du skapar i Adobe® Flash®-utvecklingsmiljön. Den har alla beteenden och funktioner som finns i visningsobjekt, men med ytterligare egenskaper och metoder för styrning av dess tidslinje.

Grunderna för filmklipp

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Filmklipp är nyckelelement för dem som vill skapa animerat innehåll med Flash-utvecklingsverktyget och vill styra innehållet med ActionScript. När du skapar en filmklippssymbol i Flash, läggs symbolen i Flash-dokumentets bibliotek. Den här symbolen blir som standard en instans av klassen MovieClip, och som sådan har den också klassen MovieClips egenskaper och metoder.

När en instans av en filmklippssymbol placeras på scenen, går filmklippsförloppet automatiskt igenom sin tidslinje (om den har fler än en bildruta) om inte uppspelningen har ändrats med ett ActionScript. Det är tidslinjen som urskiljer klassen MovieClip så att du kan skapa animering genom rörelse- eller figurinterpoleringar med Flash-utvecklingsverktyget. I motsats till ett visningsobjekt som är en instans av klassen Sprite, kan du endast skapa animering genom att ändra programmeringen för objektvärdena.

I tidigare versioner av ActionScript var klassen MovieClip basklassen för alla instanser på scenen. I ActionScript 3.0 är ett filmklipp endast ett av många visningsobjekt som kan visas på scenen. Om en tidslinje inte är nödvändig för funktionen i ett visningsobjekt, kan återgivningsprestandan eventuellt förbättras genom användning av klasserna Shape eller Sprite i stället för klassen MovieClip. Mer information om användning av lämpligt visningsobjekt för en uppgift finns i ”Välja underklassen DisplayObject” på sidan 172.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer om filmklipp:

AVM1 SWF En SWF-fil som skapats med ActionScript 1.0 eller ActionScript 2.0 har vanligtvis gjorts för Flash Player 8 eller tidigare versioner.

AVM2 SWF En SWF-fil som skapats med ActionScript 3.0 för Adobe Flash Player 9 eller senare eller Adobe AIR.

Extern SWF En SWF-fil som skapats separat från SWF-filen i projektet och är avsedd att läsas in till SWF-projektfilen och spelas upp i den SWF-filen.

Bildruta Den minsta indelningen av tid på tidslinjen. Liksom en filmremsa med rörliga bilder är varje bildruta som en ögonblicksbild av animeringen, och när bildrutorna spelas upp snabbt i sekvens skapas en animeringseffekt.

Tidslinje Den metaforiska representationen av serierna med bildrutor som utgör filmklippets animeringssekvens. Tidslinjen i ett MovieClip-objekt motsvarar tidslinjen i Flash-utvecklingsverktyget.

Spelhuvud En markör som identifierar vad (vilken bildruta) på tidslinjen som visas vid ett visst tillfälle.

Arbeta med MovieClip-objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du publicerar en SWF-fil konverterar Flash alla instanser av filmklippssymbolerna på scenen till MovieClip-objekt. Du kan göra en filmklippssymbol tillgänglig för ActionScript genom att ge den ett instansnamn i fältet Instansnamn i egenskapsinspektören. När SWF-filen har skapats genererar Flash koden som skapar MovieClip-instansen på scenen och deklarerar en variabel med instansnamnet. Om du har namngett filmklipp som är inkapslade i andra namngivna filmklipp, behandlas dessa underordnade filmklipp som egenskaper för det överordnade filmklippet. Du kan nå det underordnade filmklippet med punktsyntax. Om ett filmklipp med instansnamnet `childClip` är inkapslat i ett annat klipp med instansnamnet `parentClip` kan du få det underordnade klippets tidslinjeanimering att spelas upp genom att anropa följande kod:

```
parentClip.childClip.play();
```

Obs! *: Underordnade instanser som placeras på scenen i Flash-utvecklingsverktyget kan inte användas av kod i konstruktorn i en överordnad instans, eftersom de ännu inte har skapats vid det tillfället i kodkörningen. Innan den underordnade instansen används måste den överordnade instansen antingen skapa den underordade instansen med kod eller fördröja användandet till en återanropsfunktion som avlyssnar att den underordnade instansen skickar sin* `Event.ADDED_TO_STAGE`*-händelse.*

En del äldre metoder och egenskaper för klassen ActionScript 2.0 MovieClip är fortfarande desamma medan andra har ändrats. Alla egenskaper med understreck som prefix har bytt namn. Egenskaperna `_width` och `_height` har ändrats till `width` och `height` medan `_xscale` och `_yscale` har ändrats till `scaleX` och `scaleY`. En fullständig lista över egenskaperna och metoderna för klassen MovieClip finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Styra uppspelning av filmklipp

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash används metaforen tidslinje för förmedling av animering eller en ändring av läget. Alla visuella element som använder en tidslinje måste antingen vara ett MovieClip-objekt eller komma från klassen MovieClip. Ett ActionScript kan instruera ett filmklipp att stoppa, spela upp eller flytta till en annan punkt på tidslinjen men det kan inte användas för att dynamiskt skapa en tidslinje eller lägga till innehåll i specifika bildrutor; det går endast att göra med Flash-utvecklingsverktyget.

När ett MovieClip spelas upp, förflyttas det längs sin tidslinje med en hastighet som anges av bildrutehastigheten i SWF-filen. Alternativt kan du åsidosätta inställningen genom att ange egenskapen `Stage.frameRate` i ActionScript.

Spela upp filmklipp och stoppa uppspelningen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoderna `play()` och `stop()` får du baskontroll över ett filmklipp längs dess tidslinje. Antag att du t.ex. har en filmklippssymbol på scenen som innehåller en animering av en cykel som förflyttas över skärmen och har instansnamnet `bicycle`. Om följande kod har lagts till i en nyckelbildruta på huvudtidslinjen,

```
bicycle.stop();
```

kommer cykeln inte att röra sig (animeringen spelas inte upp). Cykelns rörelse kunde inte starta på grund av någon annan användaråtgärd. Om du t.ex. har en knapp med namnet `startButton` och följande kod finns i en nyckelbildruta på huvudtidslinjen, spelas animeringen upp när du klickar på knappen:

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

Snabbspolning framåt och bakåt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoderna `play()` och `stop()` är inte det enda sättet att styra uppspelningen i ett filmklipp. Du kan också flytta spelhuvudet framåt och bakåt längs tidslinjen manuellt med metoderna `nextFrame()` och `prevFrame()`. När någon av de här metoderna anropas, stoppas uppspelningen och spelhuvudet flyttas en bildruta framåt respektive bakåt.

Metoden `play()` är jämförbar med att anropa `nextFrame()` varje gång filmklippsobjektets `enterFrame`-händelse aktiveras. Längs de här linjerna kan du få filmklippet `bicycle` att spelas upp baklänges genom att skapa en händelseavlyssnare för `enterFrame`-händelsen och instruera `bicycle` att gå till den föregående bildrutan i avlyssningsfunktionen enligt följande:

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

I en vanlig uppspelning, om filmklippet innehåller fler än en bildruta, kommer det att upprepas oavbrutet under uppspelningen; d.v.s. återgå till bildruta 1 om det fortsätter förbi sin sista bildruta. När du använder `prevFrame()` eller `nextFrame()`, sker inte det här automatiskt (anrop av `prevFrame()` när spelhuvudet är på bildruta 1 flyttar inte spelhuvudet till den sista bildrutan). Villkoret `if` i exemplet ovan kontrollerar om spelhuvudet har gått bakåt till den första bildrutan och flyttar spelhuvudet framåt till dess sista bildruta och skapar på så sätt effektivt en kontinuerlig slinga som spelar upp filmklippet baklänges.

Hoppa till en annan bildruta och använda bildruteetiketter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att skicka ett filmklipp till en ny bildruta är enkelt. Genom att anropa antingen `gotoAndPlay()` eller `gotoAndStop()` får du filmklippet att hoppa till det bildrutenummer som anges i en parameter. Alternativt kan du skicka en sträng som matchar namnet på en bildruteetikett. Alla bildrutor i tidslinjen kan tilldelas en etikett. Det gör du genom att välja en bildruta på tidslinjen och sedan ange ett namn i fältet Bildruteetikett i egenskapsinspektören.

Fördelarna med att använda bildruteetiketter i stället för nummer blir särskilt uppenbart när du skapar komplexa filmklipp. När antalet bildrutor, lager och interpoleringar blir stort i en animering, bör du överväga att sätta etiketter med förklarande beskrivningar på viktiga bildrutor som representerar ändrat beteende i filmklippet (t.ex. ”av”, ”gå” eller ”springa”). Det förbättrar kodens läsbarhet och ger även flexibilitet eftersom ActionScript-anrop som går till en bildruteetikett är pekare till en enstaka referens, etiketten, i stället för ett visst bildrutenummer. Om du senare bestämmer dig för att flytta ett visst segment av animeringen till en annan bildruta, behöver du inte ändra ActionScript-koden så länge som du behåller samma etikett för bildrutorna på den nya platsen.

Använd klassen FrameLabel i ActionScript 3.0 för representation av bildruteetiketter i koden. Varje instans av den här klassen representerar en enstaka bildruteetikett och har en `name`-egenskap som representerar namnet på bildruteetiketten enligt specifikationen i egenskapsinspektören och en `frame`-egenskap som representerar bildrutenumret för bildrutan där etiketten har placerats på tidslinjen.

För att komma till FrameLabel-instansen som hör ihop med en filmklippsinstans, ingår det två egenskaper i klassen MovieClip som returnerar FrameLabel-objekt direkt. Egenskapen `currentLabels` returnerar en array som består av alla FrameLabel-objekt som finns på en tidslinje i ett filmklipp. Egenskapen `currentLabel` returnerar en sträng som innehåller namnet på bildruteetiketten som påträffades senast längs tidslinjen.

Antag att du höll på att skapa ett filmklipp med namnet `robot` och har angett en etikett för de olika lägena i animationen. Du kan ange ett villkor som kontrollerar egenskapen `currentLabel` för att komma till det aktuella läget i `robot` enligt följande kod:

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

I Flash Player 11.3 och AIR 3.3 har händelsen `frameLabel` lagts till i klassen FrameLabel. Du kan tilldela en händelsehanterare till den FrameLabel-instans som representerar en bildruteetikett. Händelsen skickas när spelhuvudet når bildrutan.

I följande exempel skapas en FrameLabel-instans för den andra bildruteetiketten i arrayen med bildruteetiketter för MovieClip. Sedan registreras en händelsehanterare för händelsen `frameLabel`:

```
var myFrameLabel:FrameLabel = robot.currentLabels[1];
myFrameLabel.addEventListener(Event.FRAME_LABEL, onFrameLabel);

function onFrameLabel(e:Event):void {
    //do something
}
```

Arbeta med scener

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash-utvecklingsmiljön kan du använda scener för att avgränsa en serie tidslinjer som SWF-filen kommer att gå igenom. När du använder den andra parametern för metoderna `gotoAndPlay()` eller `gotoAndStop()` kan du ange till vilken scen spelhuvudet ska skickas. Alla FLA-filer startar endast med den initiala scenen men du kan skapa nya scener.

Att använda scener är inte alltid den bästa lösningen eftersom det finns några nackdelar med dem. Ett Flash-dokument som innehåller flera scener kan vara svårt att underhålla, särskilt i en miljö med flera författare. Flera scener kan också vara ineffektivt avseende bandbredd, eftersom publiceringsprocessen sammanfogar alla scener till en enda tidslinje. Det här förorsakar en progressiv nedladdning av alla scener även om de inte spelas upp. På grund av detta bör flera scener inte användas förutom då flera långa tidslinjebaserade animeringar ska organiseras.

Egenskapen `scenes` i klassen MovieClip returnerar en array av Scene-objekt som representerar alla scener i SWF-filen. Egenskapen `currentScene` returnerar ett Scene-objekt som representerar den aktuella scenen som spelas upp.

Klassen Scene har flera egenskaper som ger information om scenen. Egenskapen `labels` returnerar en array av FrameLabel-objekt som representerar bildruteetiketterna på den scenen. Egenskapen `name` returnerar scenens namn som en sträng. Egenskapen `numFrames` returnerar ett int-objekt som representerar det totala antalet bildrutor på scenen.

Skapa MovieClip-objekt med ActionScript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett sätt att lägga till innehåll på skärmen i Flash är genom att dra resurser från biblioteket till scenen, men det finns andra arbetssätt. I komplexa projekt föredrar erfarna utvecklare vanligtvis att skapa filmklipp med programkod. Det finns flera fördelar med det sättet: Enklare återanvändning av kod, snabbare kompileringshastighet och mer avancerade ändringar som är tillgängliga endast i ActionScript.

Visningslist-API i ActionScript 3.0 effektiviserar processen att dynamiskt skapa MovieClip-objekt. Möjligheten att skapa en MovieClip-instans direkt, skilt från processen att lägga till den i visningslistan, ger flexibilitet och enkelhet utan avkall på kontrollen.

I ActionScript 3.0, när du skapar en filmklippsinstans (eller något annat visningsobjekt) med kod, är den inte synlig på skärmen innan den har lagts till i visningslistan genom anrop av metoden `addChild()` eller `addChildAt()` i en visningsobjektbehållare. Det här gör att du kan skapa ett filmklipp, ange dess egenskaper och även anropa metoder innan den återges på skärmen. Mer information om hur du använder visningslistan finns i ”Arbeta med behållare för visningsobjekt” på sidan 159.

Exportera biblioteksymboler för ActionScript.

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som standard kan instanser av filmklippssymboler i ett Flash-dokumentbibliotek inte skapas dynamiskt (d.v.s. skapas endast med ActionScript). Det beror på att varje symbol som exporteras för användning i ActionScript ökar storleken på SWF-filen och det är känt att en del symboler kanske inte är avsedda för användning på scenen. Därför måste du ange att symbolen ska exporteras för ActionScript, för att tillåta den i ActionScript.

Så här exporterar du en symbol för ActionScript:

1 Välj en symbol i bibliotekspanelen och öppna dialogrutan symbolegenskaper.

2 Aktivera avancerade inställningar om det behövs.

3 Markera rutan Exportera för ActionScript i länkningsdelen.

   Det aktiverar fälten Klass och Basklass.

   Som standard fylls fältet Klass i med symbolnamnet utan mellanslag (en symbol med namnet ”Tree House” kommer att heta ”TreeHouse”). Om du vill ange att symbolen ska använda en anpassad klass för sitt beteende, anger du fullständigt namn på klassen inklusive dess paket i det här fältet. Om du vill ha möjlighet att skapa instanser av symbolen i ActionScript, men inte behöver lägga till ytterligare beteenden, kan du lämna klassnamnet som det är.

   Standardvärdet för fältet Basklass är `flash.display.MovieClip`. Om du vill att symbolen ska utöka funktionerna för någon annan anpassad klass, kan du ange den klassens namn i stället så länge som klassen utökar klassen Sprite (eller klassen MovieClip).

4 Klicka på OK för att spara ändringarna.

   I det här läget visas en varning om Flash inte kan finna en länkad SWC-fil eller en extern ActionScript-fil med en definition för den specificerade klassen (t.ex. om du inte behövde lägga till ytterligare beteende för symbolen):

   Det gick inte att hitta en definition för den här klassen i klassökvägen. En definition kommer därför att skapas automatiskt i SWF-filen vid exporten.

   Du kan bortse från den här varningen om bibliotekssymbolen inte kräver en unik funktion förutom funktionerna i klassen MovieClip.

Om du inte anger en klass för symbolen, kommer Flash att skapa en klass motsvarande den här:

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Om du vill lägga till extra ActionScript-funktioner i symbolen, lägger du till lämpliga egenskaper och metoder i kodstrukturen nedan. Antag att du t.ex. har en filmklippssymbol som innehåller en cirkel 50 pixlar bred och 50 pixlar hög och symbolen har specificerats för att exporteras för ActionScript med klassnamnet Circle. Följande kod utökar klassen MovieClip när den placeras i en Circle.as-fil och symbolen får dessutom metoderna `getArea()` och `getCircumference()`:

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

Följande kod som placerats i en nyckelbildruta i bildruta 1 i Flash-dokumentet, skapar en instans av symbolen och visar den på skärmen:

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

I den här koden visas ActionScript-baserad instansiering som ett alternativ till att dra och släppa individuella resurser på scenen. Den skapar en cirkel som har alla egenskaper från ett filmklipp och den har också anpassade metoder som definierats i klassen Circle. Det här är ett mycket grundläggande exempel; biblioteksymbolen kan specificera ett stort antal egenskaper och metoder i sin klass.

ActionScript-baserad instansiering är kraftfull eftersom den gör det möjligt att dynamiskt skapa ett stort antal instanser som skulle vara krävande att skapa manuellt. Den är också flexibel eftersom du kan anpassa varje instans egenskap när den har skapats. Du kan få en uppfattning om båda fördelarna om du använder en slinga för att dynamiskt skapa flera Circle-instanser. Med symbolen Circle och klassen som beskrevs tidigare i Flash-dokumentets bibliotek, placerar du följande kod i en nyckelbildruta i bildruta 1:

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Här visas hur du kan snabbt kan skapa och anpassa flera instanser av en symbol utan kod. Varje instans placeras baserat på aktuellt värde inne i slingan och varje instans ges en slumpvis vald färg genom angivande av dess `transform`-egenskap (vilken Circle ärver genom utökning av klassen MovieClip).

Läsa in extern SWF-fil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 läses SWF-filer in med klassen Loader. För att läsa in en extern SWF-fil, måste ActionScript göra fyra saker:

1. Skapa ett nytt URLRequest-objekt med filens URL.
2. Skapa ett nytt Loader-objekt.
3. Anropa `load()`-metoden för Loader-objektet och överföra URLRequest-instansen som en parameter.
4. Anropa `addChild()`-metoden för en visningsobjektbehållare (t.ex. som huvudtidslinjen i ett Flash-dokument) för att lägga till Loader-instansen i visningslistan.

Koden bör slutligen se ut så här:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Samma kod kan användas för att läsa in en extern bildfil som en JPEG-, GIF- eller PNG-bild genom att ange bildfilens URL i stället för SWF-filens URL. En SWF-fil kan, till skillnad från en bildfil, innehålla ActionScript. När du läser in en extern SWF-fil måste därför både den inläsande SWF-filen och SWF-filen som läses in finnas i samma säkerhetssandlåda om Flash Player eller AIR spelar SWF-filen, även om processen att läsa in en SWF-fil kan vara identisk med inläsningen av en bild, om du planerar att använda ActionScript för kommunikation med den externa SWF-filen. Dessutom, om den externa SWF-filen innehåller klasser som delar samma namnutrymme som klasserna i den inläsande SWF-filen, kanske du måste skapa en ny programdomän för den inlästa SWF-filen för att undvika namnutrymmeskonflikter. Mer information om säkerhet och programdomäner finns i ”Arbeta med programdomäner” på sidan 148 och ”Inläsning av innehåll” på sidan 1064.

När den externa SWF-filen har lästs in, kan den nås via egenskapen `Loader.content`. Om den externa SWF-filen publiceras för ActionScript 3.0, kommer den att vara antingen ett filmklipp eller en sprite, beroende på vilken klass den utökar.

Det finns ett antal skillnader mellan inläsning av en SWF-fil i Adobe AIR for iOS jämfört med andra plattformar. Du hittar mer information i ”Läsa in SWF-filer i AIR for iOS” på sidan 201.

Beaktanden för inläsning av äldre SWF-fil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om den externa SWF-filen har publicerats med en äldre version av ActionScript, måste du ta hänsyn till viktiga begränsningar. Till skillnad från en ActionScript 3.0-SWF-fil som körs på AVM2 (ActionScript Virtual Machine 2), körs en SWF-fil som publicerats för ActionScript 1.0 eller 2.0 på AVM1 (ActionScript Virtual Machine 1).

Det finns viktiga skillnader mellan inläsning av en SWF-fil i ActionScript 1.0 eller 2.0-format till ActionScript 3.0-format (jämfört med inläsning av en SWF-fil i ActionScript 3.0). Flash Player är fullständig bakåtkompatibel med tidigare publicerat innehåll. Allt innehåll för tidigare versioner av Flash Player fungerar även i de Flash Player-versioner som har stöd för ActionScript 3.0, men med följande begränsningar:

- ActionScript 3.0-kod kan läsa in en SWF-fil som skrivits i ActionScript 1.0 eller 2.0. När en SWF-fil som skapats med ActionScript 1.0 eller 2.0 läses in felfritt är det inlästa objektet (egenskapen `Loader.content`) ett AVM1Movie-objekt. En AVM1Movie-instans är inte detsamma som en MovieClip-instans. Den är ett visningsobjekt, men till skillnad från ett filmklipp ingår inte tidslinjerelaterade metoder eller egenskaper. Den överordnade AVM2-SWF-filen kan inte komma åt egenskaperna, metoderna eller objekten i det inlästa AVM1Movie-objektet.
- SWF-filer som skrivits i ActionScript 1.0 eller 2.0 kan inte läsa in SWF-filer som skrivits i ActionScript 3.0. Det betyder att SWF-filer som redigerats i Flash 8, Flex Builder 1.5 eller tidigare versioner inte kan läsa in SWF-filer från ActionScript 3.0.

  Det enda undantaget är att en SWF-fil från ActionScript 2.0 kan ersätta sig själv med en SWF-fil från ActionScript 3.0, så länge som filen från ActionScript 2.0 inte har läst in någonting på sina nivåer. En SWF-fil från ActionScript 2.0 kan göra detta via ett anrop till `loadMovieNum()`, och överför då värdet 0 till parametern `level`.
- I allmänhet måste SWF-filer som skrivits i ActionScript 1.0 eller 2.0 flyttas om de ska fungera tillsammans med SWF-filer som skrivits i ActionScript 3.0. Låt oss anta att du skapat en mediaspelare med ActionScript 2.0. Mediaspelaren läser in blandat innehåll, som också skapats med ActionScript 2.0. Du kan inte skapa nytt innehåll i ActionScript 3.0 och läsa in det i mediaspelaren. Du måste flytta videospelaren till ActionScript 3.0.

  Om du emellertid skapar en mediaspelare i ActionScript 3.0, kan denna spelare utföra enstaka inläsningar av ditt ActionScript 2.0-innehåll.

I följande tabeller sammanfattas begränsningarna i tidigare versioner av Flash Player när det gäller att läsa in nyare innehåll och köra koder, men också begränsningarna för korsskriptning mellan SWF-filer som skrivits i olika versioner av ActionScript.

Funktioner som stöds	**Flash Player 7**	**Flash Player 8**	**Flash Player 9 och 10**
Kan läsa in SWF-filer som publicerats för	7 och tidigare	8 och tidigare	9 (eller 10) och tidigare
Innehåller detta AVM	AVM1	AVM1	AVM1 och AVM2
Kör SWF-filer som skrivits i ActionScript	1.0 och 2.0	1.0 och 2.0	1.0 och 2.0 samt 3.0

I följande tabell gäller ”Funktioner som stöds” innehåll som körs i Flash Player 9 eller senare. Innehåll som körs i Flash Player 8 eller tidigare kan läsa in, visa, köra och korsskripta bara ActionScript 1.0 och 2.0.

Funktioner som stöds	**Innehåll som har skapats i ActionScript 1.0 och 2.0**	**Innehåll som har skapats i ActionScript 3.0**
Kan läsa in innehåll och köra kod i innehåll som skapats i	Endast ActionScript 1.0 och 2.0	ActionScript 1.0 och 2.0 samt ActionScript 3.0
Kan korsskripta innehåll som skapats i	Enbart ActionScript 1.0 och 2.0 (ActionScript 3.0 via lokal anslutning)	ActionScript 1.0 och 2.0 via LocalConnection. ActionScript 3.0

Exempel på filmklipp: RuntimeAssetsExplorer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Funktionen Export för ActionScript kan vara särskilt fördelaktig för bibliotek som kan användas i fler än ett projekt. Om Flash Player eller AIR kör en SWF-fil blir symbolerna som har exporterats till ActionScript tillgängliga för alla SWF-filer i samma säkerhetssandlåda allt eftersom SWF läser in. På det här sättet kan ett enstaka Flash-dokument generera en SWF-fil som har definierats för det enda syftet att innehålla grafiska resurser. Den här tekniken är särskilt användbar i större projekt, där tecknare som arbetar med visuella resurser kan arbeta parallellt med utvecklare som skapar en SWF-wrapperfil som sedan läser in den grafiska resurs-SWF-filen under körning. Du kan använda den här metoden för att underhålla en serie versionsfiler där grafiska resurser inte är beroende av den fortlöpande programutvecklingen.

Programmet RuntimeAssetsExplorer läser in en SWF-fil som är en underklass till RuntimeAsset och du kan söka efter tillgängliga resurser i den SWF-filen. Exemplet visar följande:

- Läsa in extern SWF-fil med `Loader.load()`
- Skapa biblioteksymbol dynamiskt som exporterats för ActionScript
- ActionScript-styrning av MovieClip-uppspelning

Innan du börjar måste du kontrollera att alla SWF-filer som ska köras i Flash Player finns i samma säkerhetssandlåda. Mer information finns i ”Säkerhetssandlådor” på sidan 1049.

Om du vill ha programfilerna för det här exemplet hämtar du Flash Professional Samples. RuntimeAssetsExplorer-programfiler finns i mappen Samples/RuntimeAssetsExplorer. Programmet består av följande filer:

Fil	Beskrivning
RuntimeAssetsExample.mxml eller RuntimeAssetsExample.fla	Användargränssnittet för programmet för Flex (MXML) eller Flash (FLA).
RuntimeAssetsExample.as	Dokumentklass för Flash-programmet (FLA).
GeometricAssets.as	En exempelklass som implementerar gränssnittet RuntimeAsset.
GeometricAssets.fla	En FLA-fil länkad till klassen GeometricAssets (dokumentklassen för FLA-filen) som innehåller symboler som har exporterats för ActionScript.
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	Ett gränssnitt som definierar obligatoriska metoder som förväntas under körningen av alla resurs-SWF-filer som läses in till utforskarbehållaren.
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	Klassen för biblioteksymbolen i form av en roterande låda.
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	Klassen för biblioteksymbolen i form av en roterande stjärna.

Skapa ett biblioteksgränssnitt för körning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att utforskaren ska kunna samverka på rätt sätt med ett SWF-bibliotek, måste resursbiblioteken för körning struktureras. Vi gör det genom att skapa ett gränssnitt som liknar en klass så till vida att det är en kopia av metoder som avgränsar en förväntad struktur, men till skillnad från en klass så inkluderar den inget metodinnehåll. Med gränssnittet kan både körningsbiblioteket och utforskaren kommunicera med varandra. Varje SWF med körningsresurser som läses in i webbläsaren implementerar det här gränssnittet. Mer information om gränssnitt och hur de kan användas finns i avsnittet om gränssnitt i *Learning ActionScript 3.0*.

Gränssnittet RuntimeLibrary är enkelt; vi begär helt enkelt en funktion som kan förse utforskaren med en array av klassökvägar för symbolerna som ska exporteras och vara tillgängliga i körningsbiblioteket. Hittills har gränssnittet endast en metod: `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

Skapa resursbiblioteks-SWF-fil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Genom att definiera gränssnittet RuntimeLibrary är det möjligt att skapa flera resursbiblioteks-SWF-filer som kan läsas in till en annan SWF-fil. Fyra uppgifter ingår när du skapar ett individuellt SWF-bibliotek med resurser:

- Skapa en klass för resursbiblioteks-SWF-fil
- Skapa klasser för individuella resurser i biblioteket

- Skapa de faktiska grafikresurserna
- Associera de grafiska elementen med klasser och publicera biblioteks-SWF-filen

Skapa en klass för implementering av gränssnittet RuntimeLibrary.

Sedan skapar vi klassen GeometricAssets som implementerar gränssnittet RuntimeLibrary. Det här blir dokumentklassen för FLA. Koden för den här klassen liknar mycket den för RuntimeLibrary-gränssnittet. Skillnaden mellan dem är att i klassdefinitionen har metoden `getAssets()` ett metodinnehåll.

```
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" ];
        }
    }
}
```

Om vi ska skapa ännu ett körningsbibliotek, kan vi skapa en ny FLA baserat på en annan klass (t.ex. AnimationAssets) som har en egen `getAssets()`-implementering.

Skapa klasser för varje MovieClip-resurs

I det här exemplet utökar vi endast klassen MovieClip utan att lägga till någon funktion för de anpassade resurserna. Följande kod för AnimatingStar är jämförbar med koden för AnimatingBox:

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

Publicera biblioteket

Vi ansluter nu de MovieClip-baserade resurserna till den nya klassen genom att skapa en ny FLA och ange GeometricAssets i fältet Dokumentklass i egenskapsinspektören. För det här exemplet skapar vi två mycket enkla figurer som använder en tidslinjeinterpolering för att göra en rotering medurs över 360 bildrutor. Både `animatingBox`- och `animatingStar`-symbolerna har markerats för Export för ActionScript och i fältet Klass anges respektive klassökväg som specificerats i `getAssets()`-implementeringen. Standardbasklassen för `flash.display.MovieClip` återstår eftersom vi vill ha standard-MovieClip-metoderna som underklass.

När inställningarna för export av symbolen är klara, publicerar du FLA-filen. Nu har du skapat ditt första körningsbibliotek. Den här SWF-filen kan läsas in till en annan AVM2 SWF-fil och symbolerna AnimatingBox och AnimatingStar blir tillgängliga för den nya SWF-filen.

Läsa in biblioteket till en annan SWF-fil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den sista funktionsdelen utgörs av användargränssnittet för resursutforskaren. I det här exemplet är sökvägen till körningsbiblioteket hårdkodad som en variabel med namnet `ASSETS_PATH`. Alternativt kan du använda klassen FileReference t.ex. för att skapa ett gränssnitt som söker efter en särskild SWF-fil på hårddisken.

När körningsbiblioteket har lästs in, anropar Flash Player metoden `runtimeAssetsLoadComplete()`:

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

I den här metoden representerar variabeln rl den inlästa SWF-filen. Koden anropar `getAssets()`-metoden för den inlästa SWF-filen, hämtar listan över resurser som är tillgängliga och använder dem för att fylla i en ComboBox-komponent med en lista över tillgängliga resurser genom att anropa `populateDropDown()`-metoden. Den metoden i sin tur lagrar den fullständiga klassökvägen för varje resurs. När du klickar på Lägg till i användargränssnittet aktiveras `addAsset()`-metoden:

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

Nu hämtas klassökvägen för den resurs som för närvarande har valts i ComboBox (`assetNameCbo.selectedItem.data`) och funktionen `getDefinitionByName()` (från flash.utils-paketet) används för att hämta en faktisk referens till resursens klass för att skapa en ny instans av resursen.

Kapitel 17: Arbeta med rörelseinterpoleringar

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

I ”Animera objekt” på sidan 193 beskrivs hur du implementerar skriptad animering i ActionScript.

Här beskrivs en annan teknik för hur du skapar animering: rörelseinterpoleringar. Med den här tekniken kan du skapa rörelse genom att ställa in rörelse interaktivt i ett dokument med hjälp av Adobe® Flash® Professional. Sedan kan du använda den rörelsen i den dynamiska ActionScript-baserade animeringen vid körning.

I Flash Professional skapas det ActionScript som implementerar rörelseinterpoleringen automatiskt och blir tillgängligt för dig att kopiera och återanvända.

För att kunna skapa rörelseinterpoleringar måste du ha en licens för Flash Professional.

Fler hjälpavsnitt

fl.motion-paket

Grunderna i rörelseinterpoleringar

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Med rörelseinterpoleringar kan du på ett enkelt sätt skapa animeringar.

En rörelseinterpolering ändrar egenskaperna för visningsobjekt, till exempel position eller rotation, bildruta för bildruta. En rörelseinterpolering kan också ändra utseendet på ett visningsobjekt medan det rör sig, genom att tillämpa olika filter och andra egenskaper. Du skapar rörelseinterpoleringen interaktivt med Flash Professional, som genererar ActionScript för rörelseinterpoleringen. I Flash använder du kommandot Kopiera rörelse som ActionScript 3.0 när du vill kopiera det ActionScript som skapar rörelseinterpoleringen. Sedan kan du återanvända detta ActionScript om du vill skapa rörelse i en egen dynamisk animering under körning.

Mer information om hur du skapar en rörelseinterpolering finns i avsnittet Rörelseinterpoleringar i *Använda Flash Professional*.

Viktiga termer och begrepp

Följande är en viktig term som rör den här funktionen:

Rörelseinterpolering En konstruktion som skapar mellanliggande bildrutor för ett visningsobjekt i olika lägen och vid olika tider, vilket medför att det första läget verkar få en mjuk övergång till nästa. Används för att visa ett visningsobjekt på scenen samt få det att växa, krympa, rotera, tona ut eller ändra färg över tiden.

Kopiera skript för rörelseinterpolering i Flash

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

En interpolering skapar mellanliggande bildrutor som visar ett visningsobjekt i olika lägen i två olika bildrutor efter en tidslinje. Den skapar utseendet för bilden i den första bildrutan som sedan gör en mjuk övergång till nästa bildruta. I en rörelseinterpolering handlar förändringen i utseendet ofta om att visningsobjektet ändrar position, och på så sätt skapar en rörelse. Förutom att flytta visningsobjektet kan en rörelseinterpolering också rotera, skeva, ändra storlek på eller tillämpa filter på det.

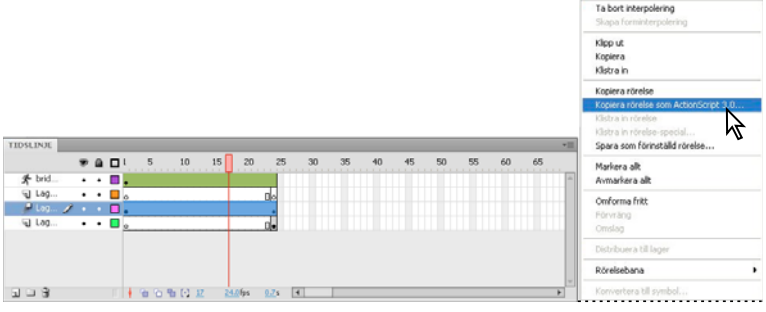
Skapa en rörelseinterpolering i Flash genom att flytta ett visningsobjekt mellan nyckelrutorna utmed tidslinjen. Flash genererar automatiskt det ActionScript som beskriver interpoleringen, som du kan kopiera och spara i en fil. Mer information om hur du skapar en rörelseinterpolering finns i avsnittet Rörelseinterpoleringar i *Använda Flash Professional*.

Du kan använda kommandot Kopiera rörelse som ActionScript 3.0 på två sätt. Det första från en interpoleringssnabbmeny på scenen:

1 Markera rörelseinterpoleringen på scenen.

2 Högerklicka (Windows) eller Control-klicka (Macintosh).

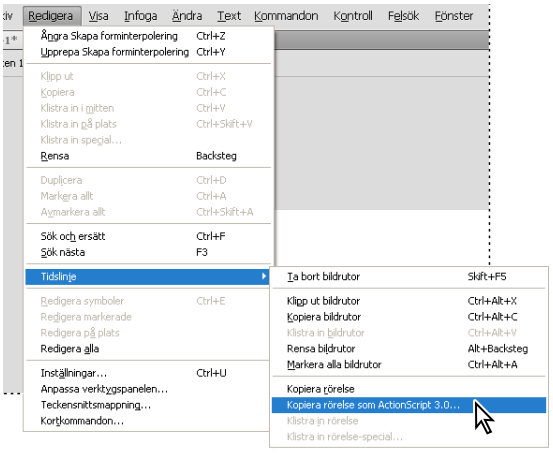
3 Välj Kopiera rörelse som ActionScript 3.0. . .



Det andra sättet är att välja kommandot direkt på Redigera-menyn i Flash:

1 Markera rörelseinterpoleringen på scenen.

2 Välj Redigera > Tidslinje > Kopiera rörelse som ActionScript 3.0.



När du har kopierat skriptet klistrar du in det i filen och sparar den.

När du har skapat en rörelseinterpolering och kopierat och sparat skriptet, kan du återanvända den som den är eller ändra den i en egen dynamisk ActionScript-baserad animering.

Infoga skript med rörelseinterpolering

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Huvudet i ActionScript-koden som du kopierar från Flash visar alla moduler som krävs för att ge stöd till rörelseinterpoleringen.

Rörelseinterpoleringsklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

De viktigaste klasserna för rörelseinterpolering är AnimatorFactory, MotionBase och Motion i paketet `fl.motion`. Du kanske behöver fler klasser beroende på vilka egenskaper som rörelseinterpoleringen manipulerar. Om rörelseinterpoleringen till exempel omformar eller roterar visningsobjektet ska du importera lämpliga `flash.geom`-klasser. Om den tillämpar filter ska du importera `flash.filter`-klasserna. I ActionScript är en rörelseinterpolering en instans av klassen Motion. Klassen Motion lagrar en animeringssekvens i en nyckelbildruta som kan användas på ett visuellt objekt. Animeringsdata omfattar position, skalning, rotering, skevning, färg, filter och övergång.

Följande ActionScript har kopierats från en rörelseinterpolering som skapades i Flash för att animera ett visningsobjekt vars instansnamn är `Symbol1_2`. Det deklarerar en variabel för ett MotionBase-objekt med namnet `__motion_Symbol1_2`. Klassen MotionBase är överordnad till klassen Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Sedan skapar skriptet Motion-objektet:

```
__motion_Symbol1_2 = new Motion();
```

Motion-objektnamn

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

I föregående fall skapades namnet `__motion_Symbol1_2` automatiskt för Motion-objektet. Prefixet `__motion_` kopplades till visningsobjektets namn. Det namn som skapades automatiskt baseras därför på instansnamnet för rörelseinterpoleringens målobjekt i Flash. Egenskapen `duration` för Motion-objektet anger det totala antalet bildrutor i rörelseinterpoleringen:

```
__motion_Symbol1_2.duration = 200;
```

Den visningsobjektinstans, vars rörelseinterpolering kopieras, får som standard automatiskt ett namn, om den inte redan har ett instansnamn.

När du återanvänder ActionScript som skapats med Flash i en egen animering kan du behålla det namn som skapades automatiskt för interpoleringen eller ange ett annat namn. Om du ändrar interpoleringens namn måste du ändra det i hela skriptet.

I Flash kan du annars tilldela ett namn till rörelseinterpoleringens målobjekt. Skapa sedan rörelseinterpoleringen och kopiera skriptet. Oavsett hur du anger namn måste du se till att varje Motion-objekt i ActionScript-koden har ett unikt namn.

Beskriva animeringen

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Metoden `addPropertyArray()` för klassen MotionBase lägger till en array med värden som beskriver alla interpolerade egenskaper.

Arrayen kan innehålla ett arrayobjekt för varje nyckelruta i rörelseinterpoleringen. Några av dessa arrayer innehåller ofta färre objekt än det totala antalet nyckelrutor i rörelseinterpoleringen. Detta inträffar när det sista värdet i arrayen inte ändras för resten av bildrutorna.

Om längden på arrayargumentet är längre än egenskapen `duration` för Motion-objektet, justerar `addPropertyArray()` värdet på egenskapen `duration` utifrån detta. Det lägger inte till nyckelrutor för egenskaper som har lagts till tidigare. De nya nyckelrutorna bevaras för de extra bildrutorna i animeringen.

Egenskaperna `x` och `y` för Motion-objektet beskriver den förändrade positionen av det interpolerade objektet allt eftersom animeringen körs. Dessa koordinater är värdena som mest troligt ändras i respektive nyckelruta, om visningsobjektets position ändras. Du kan lägga till fler rörelseegenskaper med metoden `addPropertyArray()`. Lägg till exempel till värdena `scaleX` och `scaleY` om det interpolerade objektet ändrar storlek. Lägg till värdena `scewX` och `skewY` om det skevas. Lägg till egenskapen `rotationConcat` om det roterar.

Använd metoden `addPropertyArray()` om du vill definiera följande interpoleringsegenskaper:

`x`	vågrät position för omformningspunkten för objektet i dess överordnades koordinatmodell
`y`	lodrät position för omformningspunkten för objektet i dess överordnades koordinatmodell
`z`	djupposition (z-axel) för omformningspunkten för objektet i dess överordnades koordinatmodell
`scaleX`	vågrät skala i procent för objektet utifrån dess omformningspunkt
`scaleY`	lodrät skala i procent för objektet utifrån dess omformningspunkt
`skewX`	vågrät skevningsvinkel för objektet i grader utifrån dess omformningspunkt

`skewY`	lodrät skevningsvinkel för objektet i grader utifrån dess omformningspunkt
`rotationX`	rotation av objektet runt x-axeln från dess ursprungliga orientering
`rotationY`	rotation av objektet runt y-axeln från dess ursprungliga orientering
`rotationConcat`	rotationsvärden (z-axel) för objektet i rörelsen i förhållande till tidigare orientering som tillämpas från omformningspunkten
`useRotationConcat`	Om denna anges roterar objektet när `addPropertyArray()` anger data för rörelse
`blendMode`	BlendMode-klassvärde som anger hur objektets färger blandas med grafiken under
`matrix3D`	matrix3D-egenskap om det finns någon för nyckelrutan. Används för 3D-interpoleringar. Om denna används ignoreras alla föregående omformningsegenskaper.
`rotationZ`	Objektets z-axelrotation i grader, från dess ursprungliga orientering i förhållande till den överordnade 3D-behållaren. Används för 3D-interpoleringar i stället för rotationConcat.

Vilka egenskaper som läggs till i det automatiskt genererade skriptet beror på egenskaperna som har tilldelats rörelseinterpoleringen i Flash. Du kan lägga till, flytta eller ändra några av dessa egenskaper när du anpassar en egen version av skriptet.

Följande kod tilldelar värden till egenskaperna i en rörelseinterpolering med namnet `__motion_Wheel`. I det här fallet ändrar inte det interpolerade visningsobjektet plats, utan snurrar i stället på plats genom 29 bildrutor i rörelseinterpoleringen. De värden som tilldelas arrayen `rotationConcat` definierar rotationen. Andra egenskapsvärden för rörelseinterpoleringen ändras inte.

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

I nästa exempel flyttar visningsobjektet med namnet `Leaf_1` över scenen. Dess `x`- och `y`-egenskapsarrayer innehåller olika värden för var och en av de 100 bildrutorna i animeringen. Objektet roterar dessutom runt sin `z`-axel och flyttas över scenen. Objekten i `rotationZ`-egenskapsarrayen bestämmer rotationen.

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
__motion_Symbol1_4.addPropertyArray("rotationX", [64.0361]);
__motion_Symbol1_4.addPropertyArray("rotationY", [41.9578]);
__motion_Symbol1_4.addPropertyArray("rotationZ",
    [
        -18.0336,-17.5536,-17.0736,-16.5936,-16.1136,-15.6336,-15.1536,-14.6736,
        -14.1936,-13.7136,-13.2336,-12.7536,-12.2736,-11.7936,-11.3136,-10.8336,
        -10.3536,-9.8736,-9.3936,-8.9136,-8.4336,-7.9536,-7.4736,-6.9936,-6.5136,
        -6.0336,-7.21542,-8.39723,-9.57905,-10.7609,-11.9427,-13.1245,-14.3063,
        -15.4881,-16.67,-17.8518,-19.0336,-20.2154,-21.3972,-22.5791,-23.7609,
        -24.9427,-26.1245,-27.3063,-28.4881,-29.67,-30.8518,-32.0336,-31.0771,
        -30.1206,-29.164,-28.2075,-27.251,-26.2945,-25.338,-24.3814,-23.4249,
        -22.4684,-21.5119,-20.5553,-19.5988,-18.6423,-17.6858,-16.7293,-15.7727
        -14.8162,-13.8597,-12.9032,-11.9466,-10.9901,-10.0336,-10.9427,-11.8518,
        -12.7609,-13.67,-14.5791,-15.4881,-16.3972,-17.3063,-18.2154,-19.1245,
        -20.0336,-20.9427,-21.8518,-22.7609,-23.67,-24.5791,-25.4881,-26.3972,
        -27.3063,-28.2154,-29.1245,-30.0336,-28.3193,-26.605,-24.8907,-23.1765,
        -21.4622,-19.7479,-18.0336
    ]
);
__motion_Symbol1_4.addPropertyArray("blendMode", ["normal"]);
```

Lägga till filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Om målobjektet i en rörelseinterpolering innehåller filter läggs dessa filter till med metoderna `initFilters()` och `addFilterPropertyArray()` i klassen Motion.

Initiera filterarrayen

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Metoden `initFilters()` initierar filtren. Dess första argument är en array med fullständiga klassnamn för alla filter som tillämpas på visningsobjektet. Den här arrayen med filternamn skapas utifrån filterlistan för rörelseinterpoleringen i Flash. I din kopia av skriptet kan du ta bort eller lägga till något av filtren i `flash.filters`-paketen i den här arrayen. Följande anrop initierar filterlistan för målvisningsobjektet. Det tillämpar `DropShadowFilter`, `GlowFilter` och `BevelFilter` och kopierar listan till alla nyckelrutor i Motion-objektet.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

Lägga till filter

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Metoden `addFilterPropertyArray()` beskriver egenskaperna för ett initierat filter med följande argument:

1. Det första argumentet identifierar ett filter utifrån index. Indexet refererar till den del av filnamnet i arrayen med filterklassnamn som har skickats i ett tidigare anrop till `initFilters()`.
2. Det andra argumentet är filteregenskapen som ska lagras för filtret i respektive nyckelruta.
3. Det tredje argumentet är värden på angiven filteregenskap.

Utifrån föregående anrop till `initFilters()` tilldelar följande anrop till `addFilterPropertyArray()` värdet 5 till egenskaperna `blurX` och `blurY` för `DropShadowFilter`. `DropShadowFilter` är det första (index 0) objektet i den initierade filterarrayen:

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

Nästa tre anrop tilldelar värden till kvalitets-, alfa- och färgegenskaperna för `GlowFilter`, det andra objektet (index 1) i den initierade filterarrayen:

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

Nästa fyra anrop tilldelar värden till `shadowAlpha`, `shadowColor`, `highlightAlpha` och `highlightColor` för `BevelFilter`, det tredje (index 2) objektet i den initierade filterarrayen:

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

Justera färg med ColorMatrixFilter

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

När `ColorMatrixFilter` har initierats kan du ange lämpliga `AdjustColor`-egenskaper om du vill justera intensitet, kontrast, mättnad och nyans för det interpolerade visningsobjektet. Filtret `AdjustColor` används vanligtvis när rörelseinterpoleringen skapas i Flash, och du kan finjustera det i din kopia av ActionScript-koden. Följande exempel omformar nyans och mättnad för visningsobjektet allt eftersom det rör sig.

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

Koppla en rörelseinterpolering till dess visningsobjekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare, kräver Flash CS3 eller senare

Den sista uppgiften är att koppla rörelseinterpoleringen till visningsobjektet eller -objekten som den manipulerar.

Klassen AnimatorFactory hanterar kopplingen mellan en rörelseinterpolering och dess målvisningsobjekt. Argumentet för AnimatorFactory-konstruktorn är Motion-objektet:

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

Använd metoden `addTarget()` för klassen AnimatorFactory om du vill koppla målvisningsobjektet till rörelseinterpoleringen. ActionScript som kopieras från Flash kommenteras bort från raden `addTarget()` och anger inte något instansnamn:

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

I kopian anger du visningsobjektet som ska kopplas till rörelseinterpoleringen. I följande exempel anges målen som `greenWheel` och `redWheel`:

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

Du kan koppla flera visningsobjekt till samma rörelseinterpolering med hjälp av flera anrop till `addTarget()`.

Kapitel 18: Arbeta med omvänd kinematik (IK – inverse kinematics)

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

Omvänd kinematik (IK) är en bra teknik när du skapar naturtrogen rörelse.

Med omvänd kinematik kan du skapa koordinerade rörelser i en kedja med sammankopplade delar som kallas för IK-armatur, så att delarna flyttas tillsammans på ett realistiskt sätt. Delarna i armaturen består av ben och leder. Utifrån slutpunkten i armaturen beräknas vinklarna för lederna som behövs för att nå fram till slutpunkten.

Att beräkna dessa vinklar manuellt kan vara mycket krävande. Fördelen med den här funktionen är att du kan skapa armaturer interaktivt med Adobe® Flash® Professional. Animera dem sedan med ActionScript. IK-motorn som ingår i Flash Professional utför beräkningar som beskriver armaturens rörelse. Du kan begränsa rörelserna till vissa parametrar i ActionScript-koden.

En nyhet i Flash Professional CS5-versionen av IK är begreppet benfjädring, som oftast associeras med förstklassiga animeringsprogram. Tillsammans med den nya, dynamiska fysikmotorn kan du använda den här funktionen för att konfigurera verklighetstrogna rörelser. Effekterna visas både under körning och redigering.

För att kunna skapa IK-armaturer måste du ha en licens för Flash Professional.

Fler hjälpavsnitt

fl.ik-paketet

Grunderna i omvänd kinematik

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

Med omvänd kinematik (IK) kan du skapa realistisk animering genom att koppla ihop delarna så att de flyttas i förhållande till varandra på ett verklighetstroget sätt.

Med omvänd kinematik kan du till exempel flytta ett ben till en viss position genom att röra de leder i benet som behövs för att uppnå önskad ställning. Omvänd kinematik använder ett ramverk med ben sammankopplade i en struktur som kallas för en IK-armatur. Med paketet `fl.ik` kan du skapa animeringar som påminner om naturlig rörelse. Med det kan du animera flera IK-armaturer sömlöst utan att behöva kunna något om fysiken bakom IK-algoritmerna.

Skapa IK-armaturen med stödjande ben och leder med Flash Professional. Sedan kan du använda IK-klasser för att animera dem vid körning.

Mer information om hur du skapar en IK-armatur finns i avsnittet Använda omvänd kinematik i *Använda Flash Professional*.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer om den här funktionen:

Armatur En kinematisk kedja som består av ben och leder, som används vid datoranimering för att simulera realistisk rörelse.

Ben Ett stelt segment i en armatur, som motsvarar ett ben i ett djurskelett.

Omvänd kinematik (IK) Process för att fastställa parametrarna för sammankopplade flexibla objekt som kallas för kinematisk kedja eller armatur.

Led Den plats där två ben sammankopplas för att möjliggöra benens rörelser, som motsvarar leden hos ett djur.

Fysikmotorn Ett paket med fysikrelaterade algoritmer som används för verklighetstrogna rörelser i animeringar.

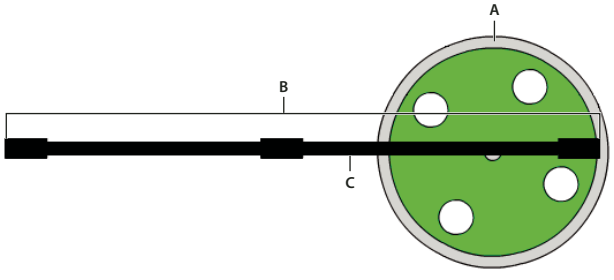
Fjädring Kvaliteten på ett ben som rör sig och reagerar när det överordnade benet rör sig och som sedan försvinner sedan successivt.

Översikt över animering av IK-armaturer (Inverse Kinematic)

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

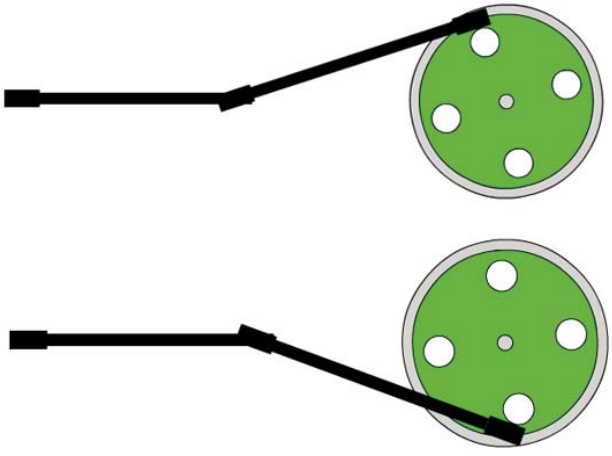
När du har skapat en IK-armatur i Flash Professional använder du `fl.ik`-klasserna för att begränsa dess rörelser, spåra dess händelser och animera den vid körning.

I bilden nedan visas ett filmklipp med namnet `Wheel`. Axeln är en instans av en IKArmature med namnet `Axle`. Klassen IKMover flyttar armaturen samtidigt som hjulet roterar. Ett IKBone, `ikBone2`, i armaturen kopplas till hjulet vid leden vid änden.



A. *Hjul* ***B.*** *Axel* ***C.*** *ikBone2*

Vid körning snurrar hjulet tillsammans med rörelseinterpoleringen `__motion_Wheel`, som behandlas i ”Beskriva animeringen” på sidan 339. Ett IKMover-objekt initieras och kontrollerar axelns rörelser. I bilden nedan visas två ögonblicksbilder av axelarmaturen som kopplas till spinnrocken vid olika bildrutor i rotationen.



ActionScript gör följande under körning:

- Hämtar information om armaturen och dess komponenter
- Instansierar ett IKMover-objekt
- Flyttar axeln i kombination med hjulets rotation

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Följande IK-klasser används för att flytta axeln:

- IKArmature: beskriver armaturen, en trädstruktur som består av ben och leder och som måste skapas med Flash Professional
- IKManager: behållarklass för alla IK-armaturer i dokumentet, som måste skapas i Flash Professional
- IKBone: ett segment i en IK-armatur
- IKJoint: en koppling mellan två IK-ben
- IKMover: initierar och kontrollerar armaturernas IK-rörelser

Fullständiga och detaljerade beskrivningar av de här klasserna finns i ik-paketet.

Hämta information om en IK-armatur (Inverse Kinematic)

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

Deklarera först variabler för armaturen, benet och leden som utgör de delar som du vill flytta.

Följande kod använder metoden `getArmatureByName()` för klassen IKManager för att tilldela värdet på axelarmaturen till IKArmature-variabeln `tree`. Axelarmaturen har redan skapats med Flash Professional.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

Följande kod använder på ett liknande sätt metoden `getBoneByName()` för klassen IKArmature för att tilldela värdet på benet `ikBone2` till IKBone-variabeln.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

Ändleden för benet `ikBone2` är den del av armaturen som kopplas till spinnrocken.

Följande rad deklarerar variabeln `endEffector` och tilldelar den till egenskapen `tailjoint` för benet `ikBone2`:

```
var endEffector:IKJoint = home.tailjoint;
```

Variabeln `pos` är en punkt som lagrar den aktuella positionen för leden `endEffector`.

```
var pos:Point = endEffector.position;
```

I det här exemplet är `pos` positionen för leden i slutet på axeln där den ansluter till hjulet. Det ursprungliga värdet på den här variabeln hämtas från egenskapen `position` för IKJoint.

Instansiera en IK Mover och begränsa dess rörelse

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

En instans av klassen IKMover flyttar axeln.

Följande rad instansierar IKMover-objektet `ik` samt skickar elementet som ska flyttas och startpunkten för rörelsen till konstruktorn:

```
var ik:IKMover = new IKMover(endEffector, pos);
```

Med egenskaperna för klassen IKMover kan du begränsa armaturens rörelse. Du kan begränsa rörelsen utifrån rörelsens avstånd, iterationer och tid.

Följande egenskapspar framtvingar dessa gränser. Paren består av ett booleskt värde, som anger om rörelsen begränsas, och ett heltal, som anger gränsen:

Boolesk egenskap	Heltalsegenskap	Begränsning
`limitByDistance:Boolean`	`distanceLimit:int`	Anger det maximala avstånd i pixlar som IK-motorn ska flytta för varje iteration.
`limitByIteration:Boolean`	`iterationLimit:int`	Anger högsta antal iterationer som IK-motorn utför för varje rörelse.
`limitByTime:Boolean`	`timeLimit:int`	Anger maxtiden (i millisekunder) under vilken IK-motorn utför rörelsen.

Som standard anges alla booleska värden till `false`, vilket innebär att rörelsen inte är begränsad om du inte uttryckligen anger ett booleskt värde som `true`. Om du vill tvinga fram en begränsning anger du relevant egenskap som `true` och anger sedan ett värde för motsvarande heltalsegenskap. Om du anger begränsningen som ett värde utan att ange motsvarande boolesk egenskap ignoreras gränsen. I så fall fortsätter IK-motorn att flytta objektet tills en annan gräns eller målpositionen för IKMover uppnås.

I följande exempel anges maxavståndet för armaturens rörelse till 0,1 pixlar per iteration. Det maximala antalet iterationer för varje rörelse anges till tio.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

Flytta en IK-armatur (Inverse Kinematic)

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

IKMover flyttar axeln i hjulets händelseavlyssnare. I varje enterFrame-händelse för hjulet beräknas en ny målposition för armaturen. Om du använder metoden `moveTo()` flyttar IKMover ändleden till målpositionen så långt bort det går inom de begränsningar som anges med egenskaperna `limitByDistance`, `limitByIteration` och `limitByTime`.

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Använda fjädring

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS5 eller senare

Omvänd kinematik i Flash Professional CS5 har stöd för benfjädring. Benfjädring kan appliceras under redigering, och benfjädringsattribut kan läggas till eller ändras under körningstid. Fjädring är en egenskap för ett ben och dess leder. Den har två attribut: `IKJoint.springStrength`, som anger fjädringsmängden samt `IKJoint.springDamping`, som ökar styrkevärdets motstånd och ändrar fjädringens minskningstakt.

Fjädringsstyrka är ett procentvärde från standardvärde 0 (helt rigid) till 100 (mycket lös och kontrollerad av fysiken). Ben med fjädring reagerar på tillhörande leds rörelser. Om ingen annan översättning (rotation, x eller y) är aktiverad har fjädringsinställningarna ingen effekt.

Fjädringsdämpning är ett procentvärde från standardvärde 0 (inget motstånd) till 100 (kraftigt dämpad). Dämpning påverkar den tid som löper mellan ett bens initiala rörelse och dess återgång till viloläge.

Du kan se om det finns aktiverade fjädrar för ett IKArmature-objekt genom att kontrollera dess `IKArmature.springsEnabled` -egenskap. De övriga fjäderegenskaperna och metoderna hör till enskilda IKJoint-objekt. En led kan aktiveras för en vinkelrotation och översättning längs x- och y-axlarna. Du kan positionera en roterande leds fjädervinkel med `IKJoint.setSpringAngle` och en översättningsbar leds fjäderposition med `IKJoint.setSpringPt`.

I det här exemplet väljs ett ben efter namn och dess tailJoint identifieras. Koden testar den överordnade armaturen för att se om fjädrarna är aktiverade och anger därefter fjädringsegenskaperna för leden.

```
var arm:IKArmature = IKManager.getArmatureAt(0);
var bone:IKBone = arm.getBoneByName("c");
var joint:IKJoint = bone.tailJoint;
if (arm.springsEnabled) {
    joint.springStrength = 50; //medium spring strength
    joint.springDamping = 10; //light damping resistance
    if (joint.hasSpringAngle) {
        joint.setSpringAngle(30); //set angle for rotational spring
    }
}
```

Använda IK-händelser

Flash Player 10 och senare, Adobe AIR 1.5 och senare, kräver Flash CS4 eller senare

Med klassen IKEvent kan du skapa ett händelseobjekt som innehåller information om IK-händelser. IKEvent-informationen beskriver rörelsen som har avbrutits på grund av att angiven gräns för tid, avstånd eller iteration överskrids.

Följande kod visar en händelseavlyssnare och hanterare för spårning av tidsgränshändelser. Den här händelsehanteraren rapporterar tid, avstånd och antal iterationer samt ledegenskaper för en händelse som utlöses när tidsgränsen för IKMover överskrids.

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

Kapitel 19: Arbeta i tre dimensioner (3D)

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I Flash Player- och AIR-miljöerna finns två typer av stöd för 3D-grafik. Du kan använda tredimensionella visningsobjekt i Flash visningslista. Detta är lämpligt för att lägga till tredimensionella effekter i Flash-innehåll och för objekt med få polygoner. I Flash Player 11 och AIR 3, eller senare, kan du återge komplexa 3D-scener med Stage3D API.

En Stage3D-visningsport är inte ett visningsobjekt. 3D-grafik återges i stället för en visningsport som visas under Flash-visningslistan (och över eventuellt StageVideo-visningsportsplan). I stället för att använda Flash DisplayObject-klasserna för att skapa en scen, använder du ett programmeringsbart 3D-tillvägagångssätt (påminner om OpenGL och Direct3D). Detta tillvägagångssätt tar triangeldata och texturer som indata och återger scenen med skuggningsprogram som du själv tillhandhåller. Maskinvaruacceleration används när en kompatibel GPU-enhet (Graphics Processing Unit) med drivrutiner är tillgänglig på klientdatorn.

Stage3D innehåller en API på mycket låg nivå. I ett program uppmanas du att använda ett 3D-ramverk som har stöd för Stage3D. Du kan skapa egna ramverk eller använda redan tillgängliga kommersiella ramverk och ramverk med öppen källkod.

Mer information om utveckling av 3D-program med Stage3D och om tillgängliga 3D-ramverk får du om du besöker Flash Player Developer Center: Stage 3D.

Grunderna för 3D-visningsobjekt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Den största skillnaden mellan ett tvådimensionellt (2D) objekt och ett tredimensionellt (3D) objekt som projiceras på en tvådimensionell skärm, är att objektet får en tredje dimension. Tack vare den tredje dimensionen kan objektet flyttas mot och bort från användarens vypunkt.

När du uttryckligen ställer in egenskapen `z` för ett visningsobjekt till ett numeriskt värde, skapar objektet automatiskt en 3D-omformningsmatris. Du kan ändra den här matrisen om du vill ändra 3D-omformningsinställningarna för objektet.

Även 3D-rotation skiljer sig från 2D-rotation. I 2D är axelrotationen alltid vinkelrät mot x/y-planet, med andra ord utmed z-axeln. I 3D kan axelrotationen ske runt såväl x- som y- och z-axeln. Om du ställer in rotations- och skalningsegenskaper för ett visningsobjekt kan det flytta i 3D-rymden.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kan stöta på när du programmerar tredimensionell grafik:

Perspektiv I ett 2D-plan är detta en representation av parallella linjer som sammanstrålar vid flyktpunkten och ger en illusion av djup och avstånd.

Projektion Skapa en 2D-bild av ett objekt med högre dimension. 3D-projektion mappar 3D-punkter till ett 2D-plan.

Rotation Ändra ett objekts orientering (och ofta position) genom att flytta alla punkter som ingår i objektet i en cirkelformad rörelse.

Omformning Ändra 3D-punkter eller ställa in punkter genom översättning, rotation, skalning, skevning eller en kombination av dessa åtgärder.

Översättning Ändra position för ett objekt genom att flytta alla punkter som ingår i objektet med samma mängd i samma riktning.

Flyktpunkt Punkt där parallella linjer som avlägsnar sig verkar sammanstråla när de visas i ett linjärt perspektiv.

Vektor En 3D-vektor representerar en punkt eller plats i ett tredimensionellt rum med hjälp de av kartesiska koordinaterna x, y och z.

Hörn En hörnpunkt.

Texturerat nät En punkt som definierar ett objekt i 3D-rymden.

UV-mappning Ett sätt att använda en textur eller bitmapp på en 3D-yta. UV-mappning tilldelar värden till koordinater i en bild som procent av den vågräta (U) axeln och den lodräta (V) axeln.

T-värde Skalningsfaktorn som bestämmer storleken på ett 3D-objekt allt eftersom objektet flyttas mot, eller bort från, den aktuella vypunkten.

Culling Återge, eller inte återge, ytor med särskild vridning. Med culling kan du dölja ytor som inte syns från den aktuella vypunkten.

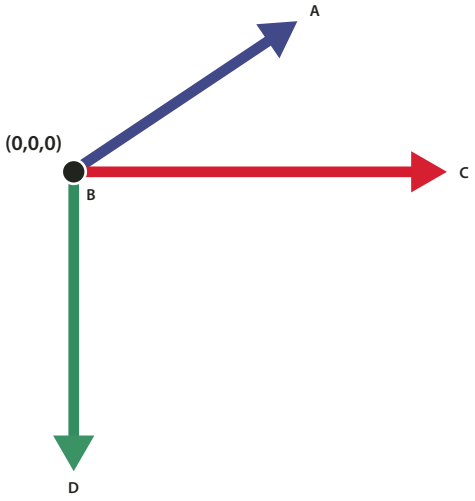
Om 3D-visningsobjekt i Flash Player och i AIR-miljön

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I tidigare versioner än Flash Player 10 och Adobe AIR 1.5 har visningsobjekt två egenskaper, `x` och `y`, för placering på ett 2D-plan. Från och med Flash Player 10 och Adobe AIR 1.5 har alla ActionScript-visningsobjekt en `z`-egenskap som du kan använda för att placera objekt på z-axeln, som normalt sett används för att ange djup eller avstånd.

3D-effekter stöds från och med Flash Player 10 och Adobe AIR 1.5. Visningsobjekt är emellertid till sin natur platta. Varje visningsobjekt, till exempel ett MovieClip-objekt eller ett Sprite-objekt, återger sig själva i två dimensioner, på ett enda plan. Med 3D-funktionerna kan du montera, flytta, rotera och på annat sätt omforma dessa platta objekt i alla tre dimensioner. Med dem kan du också hantera 3D-punkter och konvertera dem till tvådimensionella x- och y-koordinater, så att du kan projicera 3D-objekt i en 2D-vy. Du kan simulera många olika typer av 3D-upplevelser med dessa funktioner.

Det 3D-koordinatsystem som används i ActionScript skiljer sig något från andra system. När du använder 2D-koordinater i ActionScript ökar värdet på x allt eftersom du flyttar åt höger utmed x-axeln, och värdet på y ökar allt eftersom du flyttar nedåt utmed y-axeln. 3D-koordinatsystemet använder fortfarande dessa regler men tillför en z-axel vars värde ökar allt eftersom du flyttar bort från vypunkten.



De positiva riktningarna för x-, y- och z-axlarna i 3D-koordinatsystemet i ActionScript.
A. *+Z-axel* ***B.*** *Origo* ***C.*** *+X-axel* ***D.*** *+Y-axel*

Obs! *Tänk på att Flash Player och AIR alltid representerar 3D i lager. Det innebär att om objekt A finns framför objekt B i visningslistan, återges alltid A framför B oavsett de två objektens z-axelvärden. Du löser den här konflikten mellan ordningen i visningslistan och ordningen utmed z-axeln genom att använda metoden* `transform.getRelativeMatrix3D()` *för att spara och sedan ändra ordning på lagren för 3D-visningsobjekten. Mer information finns i avsnittet "Ändra visningsordning med hjälp av Matrix3D-objekt" på sidan 362.*

Följande ActionScript-klasser har stöd för de nya 3D-relaterade funktionerna:

1. Klassen flash.display.DisplayObject innehåller egenskapen `z` och nya rotations- och skalningsegenskaper för manipulering av visningsobjekt i 3D-rymden. Med metoden `DisplayObject.local3DToGlobal()` kan du på ett enkelt sätt projicera 3D-geometri i ett 2D-plan.
2. Klassen flash.geom.Vector3D kan användas som en datastruktur för hantering av 3D-punkter. Den har också stöd för vektormatematik.
3. Klassen flash.geom.Matrix3D har stöd för komplexa omformningar av 3D-geometri, till exempel rotation, skalning och översättning.
4. Klassen flash.geom.PerspectiveProjection styr parametrarna för mappning av 3D-geometri i en 2D-vy.

Det finns två olika sätt att simulera 3D-bilder i ActionScript:

1. Ordna och animera planobjekt i 3D-rymden. På detta sätt animerar du visningsobjekt med hjälp av x-, y- och z-egenskaperna för visningsobjekten, eller anger rotations- och skalningsegenskaper med hjälp av klassen DisplayObject. Du kan skapa mer komplexa rörelser med objektet DisplayObject.transform.matrix3D. Objektet DisplayObject.transform.perspectiveProjection anpassar hur visningsobjekten ritas i 3D-perspektiv. Använd det här sättet när du vill animera 3D-objekt som i huvudsak består av plan. Exempel på det här angreppssättet är 3D-bildgallerier eller 2D-animeringsobjekt som ordnas i 3D-rymden.
2. Generera 2D-trianglar från 3D-geometri, och återge dessa trianglar med texturer. Om du vill använda det här sättet måste du först definiera och hantera data om 3D-objekten och sedan konvertera dessa data till 2D-trianglar för återgivning. Bitmappstexturer kan mappas till dessa trianglar, och sedan ritas trianglarna i ett grafiskt objekt med metoden `Graphics.drawTriangles()`. Exempel på det här angreppssättet är att läsa in 3D-modelldata från en fil och återge modellen på skärmen, eller generera och rita 3D-terräng som triangelnät.

Skapa och flytta 3D-visningsobjekt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Om du vill konvertera ett 2D-visningsobjekt till ett 3D-visningsobjekt anger du explicit dess `z`-egenskap till ett numeriskt värde. När du tilldelar ett värde till `z`-egenskapen, skapas ett nytt Transform-objekt för visningsobjektet. När du ställer in egenskapen `DisplayObject.rotationX` eller `DisplayObject.rotationY` skapas också ett nytt Transform-objekt. Transform-objektet innehåller en `Matrix3D`-egenskap som styr hur visningsobjektet visas i 3D-rymden.

Följande kod ställer in koordinaterna för ett visningsobjekt med namnet ”leaf”:

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Dessa värden, samt egenskaper som härleds från dessa värden, visas i `matrix3D`-egenskapen för bladets Transform-objekt:

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

Mer information om egenskaperna för Transform-objektet finns i avsnittet om Transform-klassen. Mer information om egenskaperna för Matrix3D-objektet finns i avsnittet om Matrix3D-klassen.

Flytta ett objekt i 3D-rymden

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan flytta ett objekt i 3D-rymden genom att ändra värdena på dess `x-, y- och z`-egenskaper. När du ändrar värdet på `z`-egenskapen verkar det som om objektet flyttas närmare eller längre bort från vypunkten.

Följande kod flyttar två ellipser bakåt och framåt utmed deras z-axlar genom att ändra värdet på `z`-egenskaperna som svar på en händelse. `ellipse2` flyttar snabbare än `ellipse1`: `z`-egenskapen för ellipse2 ökar med multipeln 20 vid varje Frame-händelse medan `z`-egenskapen för `ellipse1` ökar med multipeln 10:

```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

Rotera ett objekt i 3D-rymden

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan rotera ett objekt på tre olika sätt, beroende på hur du ställer in objektets rotationsegenskaper: `rotationX`, `rotationY` och `rotationZ`.

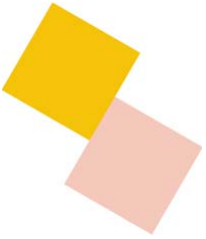
I bilden nedan visas två fyrkanter som inte roterar:

I nästa bild visas de två fyrkanterna när du ökar `rotationY`-egenskapen för fyrkanternas behållare för att rotera dem vid y-axeln. Om du roterar behållaren, eller det överordnade visningsobjektet, roteras båda fyrkanterna:

```
container.rotationY += 10;
```

I nästa bild visas vad som händer när du ställer in `rotationX`-egenskapen för fyrkanternas behållare. Då roteras fyrkanterna vid x-axeln.

I nästa bild visas vad som händer när du ökar `rotationZ`-egenskapen för fyrkanternas behållare. Då roteras de vid z-axeln.



Ett visningsobjekt kan flytta och rotera samtidigt i 3D-ytan.

Projicera 3D-objekt i en 2D-vy

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med klassen PerspectiveProjection i `flash.geom`-paketet kan du på ett enkelt sätt få ett elementärt perspektiv för att flytta visningsobjekt i 3D-rymden.

Om du inte uttryckligen skapar en perspektivprojektion för 3D-rymden, använder 3D-motorn standard-PerspectiveProjection-objektet som finns i roten och det sprids till alla underordnade.

Följande tre egenskaper definierar hur ett PerspectiveProjection-objekt visas i 3D-rymden:

- `fieldOfView`
- `projectionCenter`
- `focalLength`

Om du ändrar värdet på `fieldOfView` ändras värdet på `focalLength` automatiskt, och vice-versa, eftersom de är beroende av varandra.

Formeln som används för att beräkna `focalLength` med givet `fieldOfView`-värde är:

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

Vanligtvis ändrar du `fieldOfView`-egenskapen uttryckligen.

Synfält

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Genom att manipulera egenskapen `fieldOfView` för klassen PerspectiveProjection kan du göra så att ett 3D-visningsobjekt som närmar sig betraktaren verkar vara större, och ett objekt som viker av från betraktaren verkar vara mindre.

Egenskapen `fieldOfView` anger en vinkel mellan 0 och 180 grader som bestämmer styrkan i perspektivprojektionen. Ju större värde, desto större förvrängning tillämpas på ett visningsobjekt som flyttas utmed z-axeln. Ett lågt `fieldOfView`-värde leder till en väldigt liten skalning och får det att verka som om objektet flyttar bara något bakåt i rymden. Ett högt `fieldOfView`-värde leder till större förvrängning och rörelsen verkar vara större. Det maximala värdet 179,9999... grader skapar en effekt av en extrem fisheye-lins. Det högsta värdet för `fieldOfView` är 179,9999... och det minsta värdet är 0,00001... 0 och 180 är otillåtna värden.

Projektionscentrum

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `projectionCenter` representerar flyktpunkten i perspektivprojektionen. Den används som en förskjutning från standardregistreringspunkten (0,0) i scenens övre vänstra hörn.

När ett objekt verkar flyttas längre bort från vypunkten, skevas det mot flyktpunkten för att slutligen försvinna. Föreställ dig en oändligt lång sal. När du tittar nedåt salen, sammanstrålar väggarna till en slutpunkt längst ned i salen.

Om slutpunkten finns i mitten av scenen, försvinner salen mot en punkt i mitten. Standardvärdet för egenskapen `projectionCenter` är scenens mitt. Om du till exempel vill att elementen ska visas till vänster på scenen och en 3D-yta ska visas till höger, ställer du in `projectionCenter` på en punkt som pekar till höger på scenen så att den blir flyktpunkten i 3D-visningsområdet.

Brännvidd

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `focalLength` representerar avståndet mellan vypunktens origo (0,0,0) och placeringen av visningsobjektet på z-axeln.

En lång brännvidd kan liknas vid ett teleobjektiv med en smal vy och komprimerade avstånd mellan objekten. En kort brännvidd kan liknas vid ett vidvinkelobjektiv, där du får en bred vy med stor förvrängning. En medellång brännvidd ger ungefär det som ses med det mänskliga ögat.

Egenskapen `focalLength` beräknas vanligtvis om dynamiskt under perspektivomformningen allt eftersom visningsobjektet flyttas, men du kan ställa in det uttryckligen.

Standardvärden för perspektivprojektion

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Standardobjektet för PerspectiveProjection som skapas vid roten har följande värden:

- `fieldOfView: 55`
- `perspectiveCenter: stagewidth/2, stageHeight/2`
- `focalLength: stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Dessa värden används om du inte skapar ett eget PerspectiveProjection-objekt.

Du kan instansiera ett eget PerspectiveProjection-objekt om du tänker ändra `projectionCenter` och `fieldOfView` själv. I så fall är standardvärdena för det nya objektet följande, baserat på standardstorleken 500 x 500 för en scen:

- `fieldOfView: 55`
- `perspectiveCenter: 250,250`

- `focalLength: 480.24554443359375`

Exempel: Perspektivprojektion

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I följande exempel visas hur du använder perspektivprojektion för att skapa en 3D-rymd. Här visas hur du kan ändra flyktpunkten och ändra perspektivprojektionen för rymden via egenskapen `projectionCenter`. Den här ändringen framtvingar en omberäkning av `focalLength` och `fieldOfView` med åtföljande förvrängning av 3D-rymden.

Följande exempel:

1. Skapar en sprite med namnet `center`, som en cirkel med hårkors
2. Tilldelar koordinaterna för denna `center`-sprite till egenskapen `projectionCenter` för egenskapen `perspectiveProjection` för egenskapen `transform` i roten
3. Lägger till händelseavlyssnare för mushändelser som anropar hanterare som ändrar `projectionCenter`, så att den följer platsen för `center`-objektet
4. Skapar fyra dragspelsliknande rutor som utgör väggarna i perspektivrymden

När du testar det här exemplet, ProjectionDragger.swf, drar du cirkeln runt till olika platser. Flyktpunkten följer cirkeln, och landar där du släpper den. Titta på hur rutorna som omsluter rymden sträcks ut och förvrängs när du flyttar projektionscentrum bort från scenens mitt.

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. ProjectionDragger-programfilerna finns i mappen Samples/ProjectionDragger.

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
```

```
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int
= 50, color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
```

```
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            inDrag = false;
        }
    }
}
```

För mer komplex perspektivprojektion använder du klassen Matrix3D.

Utföra komplexa 3D-omformningar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med klassen Matrix3D kan du omforma 3D-punkter inom en koordinatrymd eller mappa 3D-punkter från en koordinatrymd till en annan.

Du behöver inte förstå matrismatematiken för att kunna använda klassen Matrix3D. De flesta vanliga omformningsåtgärder kan hanteras med klassens metoder. Du behöver inte bekymra dig om att uttryckligen ställa in eller beräkna värdena för respektive element i matrisen.

När du har ställt in `z`-egenskapen för ett visningsobjekt på ett numeriskt värde, kan du hämta objektets omformningsmatris med hjälp av Matrix3D-egenskapen för visningsobjektets Transform-objekt:

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

Du kan använda metoderna för Matrix3D-objektet om du vill utföra översättning, rotation, skalning och perspektivprojektion på visningsobjektet.

Använd klassen Vector3D med dess `x`-, `y`- och `z`-egenskaper när du hanterar 3D-punkter. Den kan också representera en spatial vektor inom fysiken, som har en riktning och storlek. Med metoderna för klassen Vector3D kan du utföra vanliga beräkningar med spatiala vektorer, till exempel addition, punktprodukt och summaproduktberäkningar.

Obs! *Klassen Vector3D är inte besläktad med ActionScript-klassen Vector. Klassen Vector3D innehåller egenskaper och metoder för att definiera och manipulera 3D-punkter, medan klassen Vector stöder arrayer med typbestämda objekt.*

Skapa Matrix3D-objekt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan skapa och hämta `Matrix3D`-objekt på tre sätt:

- Använda `Matrix3D()`-konstruktormetoden för att instansiera en ny matris. `Matrix3D()`-konstruktorn använder ett `Vector`-objekt som innehåller 16 numeriska värden och placerar vart och ett av värdena i en cell i matrisen. Till exempel:

  ```
  var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
  ```

- Ange värdet för `z`-egenskapen för ett visningsobjekt. Hämta sedan omformningsmatrisen från det objektets `transform.matrix3D`-egenskap.
- Hämta Matrix3D-objektet som styr visningen av 3D-objekten på scenen genom att anropa metoden `perspectiveProjection.toMatrix3D()` för rotvisningsobjektet.

Använda flera 3D-omformningar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan tillämpa flera 3D-omformningar samtidigt med hjälp av ett Matrix3D-objekt. Om du till exempel vill rotera, skala och sedan flytta en kub, tillämpar du tre separata omformningar på varje punkt i kuben. Det är emellertid mer effektivt att beräkna flera omformningar i ett Matrix3D-objekt i förväg och sedan utföra en matrisomformning på varje punkt.

Obs! *Ordningen som matrisomformningarna utförs i har betydelse. Matrisberäkningar är inte kommutativa. Om du till exempel tillämpar en rotation följt av en översättning får du ett annat resultat än om du tillämpar samma översättning följt av samma rotation.*

I följande exempel visas två olika sätt att utföra flera 3D-omformningar.

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
```

```
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);

            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

I metoden `doTransforms()` använder det första kodblocket DisplayObject-egenskaperna för att ändra rotation, skalning och position för en rektangulär form. Det andra kodblocket använder metoderna för klassen Matrix3D för att utföra samma omformningar.

Den största fördelen med att använda `Matrix3D`-metoderna är att alla beräkningar utförs först i matrisen. Sedan tillämpas de på visningsobjektet endast en gång, när egenskapen `transform.matrix3D` har angetts. Om du ställer in DisplayObject-egenskaperna blir källkoden lite enklare att läsa. Varje gång som en rotations- eller skalningsegenskap ställs in, sker emellertid fler beräkningar som ändrar flera av visningsobjektets egenskaper.

Om koden ska tillämpa samma komplexa omformningar på visningsobjekten fler än en gång, kan du spara Matrix3D-objektet som en variabel och sedan använda den om igen.

Ändra visningsordning med hjälp av Matrix3D-objekt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Så som tidigare nämnts bestämmer visningsobjektens lagerordning i visningslistan vilken ordning lagren visas i, oavsett deras relativa z-axlar. Om animeringen omformar egenskaperna för visningsobjekten till en annan ordning än den i visningslistan, kanske betraktaren ser visningsobjekten i en lagerordning som inte motsvarar z-axelns lagerordning. Ett objekt som ska visas längre bort från betraktaren kanske därför visas framför ett objekt som är närmare betraktaren.

För att vara säker på att lagerordningen för 3D-visningsobjekten motsvarar objektens relativa djup, ska du göra så här:

1. Använd metoden `getRelativeMatrix3D()` för Transform-objektet för att hämta den relativa `z-axeln` till de underordnade 3D-visningsobjekten.
2. Ta bort objekten från visningslistan med metoden `removeChild()`.
3. Sortera visningsobjekten utifrån deras relativa z-axelvärden.
4. Lägg tillbaka de underordnade objekten i omvänd ordning i visningslistan med metoden `addChild()`.

Den här ordningen säkerställer att objekten visas enligt sina relativa z-axlar.

Följande kod framtvingar rätt visning av de sex ytorna i en 3D-ask. Den ändrar ordning på askens ytor efter en rotation:

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna finns i mappen Samples/ReorderByZ.

Använda trianglar för 3D-effekter

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I ActionScript utför du bitmappsomformningar med metoden `Graphics.drawTriangles()`, eftersom 3D-modeller representeras av en mängd trianglar i rymden. (Flash Player och AIR har emellertid inget stöd för djupbuffring, så visningsobjekten är fortfarande platta, eller tvådimensionella. Detta beskrivs i ”Om 3D-visningsobjekt i Flash Player och i AIR-miljön” på sidan 352.) Metoden `Graphics.drawTriangles()` påminner om metoden `Graphics.drawPath()` på det sätt att den använder en uppsättning koordinater för att rita en triangelbana.

Mer information om hur du använder `Graphics.drawPath()` finns i ”Rita banor” på sidan 236.

Metoden `Graphics.drawTriangles()` använder en Vector.<Number>-objekt för att ange triangelbanans punktplaceringar:

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

Den första parametern i `drawTriangles()` är den enda obligatoriska parametern: `vertices`. Den här parametern är en vektor med siffror som definierar koordinaterna som trianglarna ritas genom. Alla tre uppsättningar koordinater (sex siffror) representerar en triangelbana. Utan parametern `indices` bör längden på vektorn alltid ha faktorn sex, eftersom varje triangel måste ha tre koordinatpar (tre uppsättningar med två x/y-värden). Till exempel:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

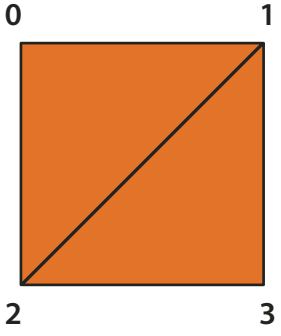
Ingen av dessa trianglar har gemensamma punkter, men om så vore fallet skulle den andra `drawTriangles()`-parametern, `indices`, kunna användas för att återanvända värdena i vektorn `vertices` för mer än en triangel.

När du använder parametern `indices` bör du vara medveten om att `indices`-värdena är punktindex, inte index som relaterar direkt till `vertices`-arrayelementen. Med andra ord, ett index i vektorn `vertices` som definieras med `indices` är i själva verket det verkliga indexet dividerat med 2. För den tredje punkten i en `vertices`-vektor kan du till exempel använda `indices`-värdet 2, även om det första numeriska värdet för den punkten startar vid vektorindex 4.

Sammanfoga till exempel två trianglar så att de delar den diagonala kanten med parametern `indices`:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Observera, att även om fyrkanten nu har ritats med hjälp av två trianglar, har bara fyra punkter angetts i `vertices`-vektorn. Om du använder `indices` kommer de två punkterna som delas av de två trianglarna att återanvändas för varje triangel. Då minskar det totala antalet hörn från 6 (12 siffror) till 4 (8 siffror):



En fyrkant som ritas med två trianglar med hjälp av parametern vertices

Den här tekniken kan vara användbar vid större triangelnät där de flesta punkter är gemensamma för flera trianglar.

Alla fyllningar kan tillämpas på trianglar. Fyllningarna används på det färdiga triangelnätet på samma sätt som på andra former.

Omforma bitmappar

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Bitmappsomformningar skapar en illusion av perspektiv eller ”textur” på ett tredimensionellt objekt. Du kan förvränga en bitmapp mot en flyktpunkt, så att bilden verkar krympa allt eftersom den flyttar mot flyktpunkten. Alternativt kan du använda en tvådimensionell bitmapp om du vill skapa en yta för ett tredimensionellt objekt, och skapa en illusion av textur eller ”ihopvikning” av det tredimensionella objektet.

En tvådimensionell yta som använder en flyktpunkt och ett tredimensionellt objekt som har vikts ihop med en bitmapp.

UV-mappning

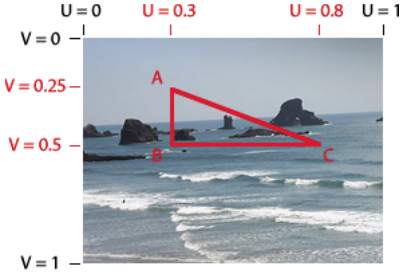
Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du har börjat arbeta med texturer vill du förmodligen också använda uvtData-parametern för `drawTriangles()`. Med den här parametern kan du ställa in UV-mappning för bitmappsfyllningar.

UV-mappning är en metod för att texturera objekt. Den utnyttjar två värden, ett U-vågrätt värde (x) och ett V-lodrätt värde (y). I stället för att baseras på pixelvärden, baseras de på procent. 0 U och 0 V innebär det övre vänstra hörnet i en bild och 1 U och 1 V det nedre högra hörnet:

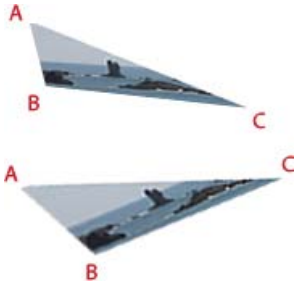
Platserna UV 0 och 1 i en bitmappsbild

Vektorerna för en triangel kan tilldelas UV-koordinater om du vill koppla dem till respektive plats i en bild:



UV-koordinaterna i ett triangulärt område i en bitmappsbild

UV-värdena fortsätter att motsvara punkterna i triangeln:



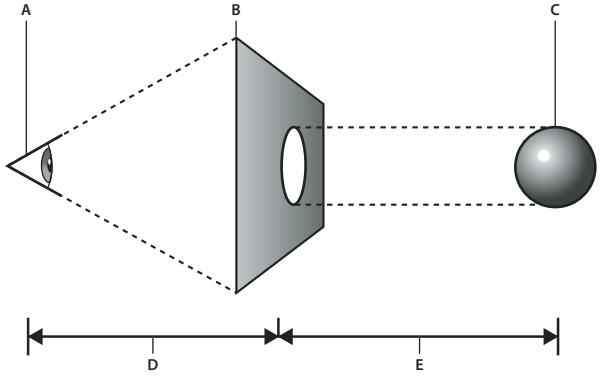
Hörnen i triangeln flyttar och bitmappen förvrängs så att UV-värdena för de enskilda punkterna inte ändras

När ActionScript 3D-omformningarna används på triangeln som hör till bitmappen, används bitmappsbilden på triangeln utifrån UV-värdena. I stället för att använda matrisberäkningar, ska du ställa in eller justera UV-värdena om du vill skapa en tredimensionell effekt.

Metoden `Graphics.drawTriangles()` kan även innehålla ytterligare information för tredimensionella omformningar: T-värdet. T-värdet i uvtData representerar 3D-perspektivet, eller mer specifikt, skalfaktorn för det tillhörande hörnet. UVT-mappning tillför perspektivkorrigering i UV-mappningen. Exempel: om ett objekt placeras i 3D-ytan bort från vypunkten så att det verkar vara 50 % av den ursprungliga storleken, blir T-värdet för det objektet 0,5. Eftersom trianglarna ritas för att representera objekt i 3D-rymden, bestämmer deras placering utmed z-axeln deras T-värden. Följande ekvation bestämmer T-värdet:

```
T = focalLength/(focalLength + z);
```

I den här ekvationen representerar focalLength en brännvidd eller beräknad ”skärmplacering” som bestämmer hur mycket perspektiv som finns i bilden.



Brännvidd och z-värde
***A.** vypunkt **B.** skärm **C.** 3D-objekt **D.** värdet på focalLength **E.** z-värde*

Värdet på T används för att skala enkla former så att de verkar vara längre bort. Det är vanligtvis detta värde som används för att konvertera 3D-punkter till 2D-punkter. När det gäller UVT-data används det också för att skala en bitmapp mellan punkterna i en triangel med perspektiv.

När du definierar UVT-värden följer T-värdet de UV-värden som har definierats för ett hörn. Med detta ”T” matchas vart tredje värde i `uvtData`-parametern (U, V och T) mot vartannat värde i `vertices`-parametern (x och y). Med enbart UV-värden blir uvtData.length == vertices.length. Om du tar med ett T-värde blir uvtData.length = 1.5*vertices.length.

I följande exempel visas ett plan som roteras i 3D-rymden med hjälp av UVT-data. I det här exemplet används en bild med namnet ocean.jpg och en hjälpklass, ImageLoader, för att läsa in bilden ocean.jpg så att den kan tilldelas BitmapData-objektet.

Här är ImageLoader-klasskällan (spara den här koden i en fil med namnet ImageLoader.as):

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

Och här är det ActionScript som använder trianglar, UV-mappning och T-värden för att göra så att bilden ser ut som om den krymper mot flyktpunkten och roterar. Spara den här koden i en fil med namnet Spinning3dOcean.as:

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);

            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
```

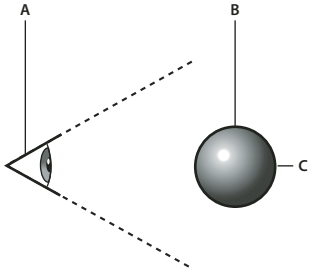
```
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

Om du vill testa det här exemplet sparar du dessa två klassfiler i samma katalog som en bild med namnet ocean.jpg. Du ser hur den ursprungliga bitmappen omformas så att det verkar som om den försvinner med avståndet och roterar i 3D-rymden.

Culling

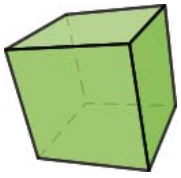
Flash Player 10 och senare, Adobe AIR 1.5 och senare

Culling är den process som bestämmer vilka ytor i ett tredimensionellt objekt som inte ska återges, eftersom de döljs från den aktuella vypunkten. I 3D-rymden är ytan på baksidan av ett tredimensionellt objekt dolt från vypunkten:



Baksidan på ett 3D-objekt är dolt från vypunkten.
A. *vypunkt* ***B.*** *3D-objekt* ***C.*** *baksidan på ett tredimensionellt objekt*

Alla trianglar återges alltid oavsett storlek, form eller position. Culling säkerställer att Flash Player eller AIR återger 3D-objekten på rätt sätt. Om du vill ha färre återgivningscykler kanske du vill att vissa trianglar hoppas över vid återgivningen. Tänk på en kub som roterar i rymden. Vid en given tidpunkt ser du aldrig mer än tre sidor av den kuben, eftersom de sidor som inte visas riktas åt ett annat håll på den andra sidan av kuben. Eftersom dessa sidor inte visas, bör de inte ritas vid återgivningen. Utan culling återges både fram- och baksidan i Flash Player och AIR.



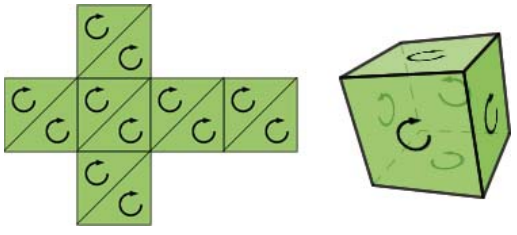
En kub har sidor som inte visas från den aktuella vypunkten

Därför har metoden `Graphics.drawTriangles()` en fjärde parameter som upprättar ett culling-värde:

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

Culling-parametern är ett värde från `TriangleCulling`-uppräkningsklassen: `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` och `TriangleCulling.NEGATIVE`. Dessa värden är beroende av triangelbanans riktning som definierar objektets yta. ActionScript-API:et för att bestämma culling förutsätter att alla utåtriktade trianglar i en 3D-form ritas med samma banriktning. När en triangelyta vänds ändras också banans riktning. Vid den punkten kan culling användas på triangeln (det vill säga ingen återgivning sker).

`TriangleCulling`-värdet `POSITIVE` tar bort trianglar med positiv banriktning (medurs). `TriangleCulling`-värdet `NEGATIVE` tar bort trianglar med negativ banriktning (moturs). När det gäller en kub, har dess sidor som riktas framåt en positiv banriktning medan sidorna som riktas bakåt har en negativ banriktning:



En utvecklad kub där banriktningen visas. Ihopvikt är baksidans banriktning den omvända.

Om du vill se hur culling fungerar börjar du med ett tidigare exempel från ”UV-mappning” på sidan 365 och ställer in culling-parametern för metoden `drawTriangles()` på `TriangleCulling.NEGATIVE`:

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Observera att baksidan på bilden inte återges när objektet roteras.

Kapitel 20: Grunderna för arbete med text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du vill visa text på skärmen i Adobe® Flash® Player eller Adobe® AIR™ använder du en instans av klassen TextField eller Flash-textmotorklasserna. Med dessa klasser kan du skapa, visa och formatera text. Du kan också använda TLF (Text Layout Framework), ett komponentbibliotek som bygger på Flash-textmotorklasserna och är utformat för att vara lätt att använda. På mobila enheter kan du använda klassen StageText för textinmatning.

Du kan fastställa specifikt innehåll för textfält eller ange textens källa och därefter textens utseende. Du kan även svara på användarhändelser, t.ex. när användaren skriver text eller klickar på en hypertext-länk.

I både klassen TextField och i klasserna Flash Text Engine tillåts du att visa och hantera text i Flash Player och AIR. Du kan använda klassen TextField för att skapa objekt för visning och inmatning. Klassen TextField innehåller grunderna för andra textbaserade komponenter, till exempel TextArea och TextInput. Du kan använda klassen TextFormat när du anger tecken- och styckeformatering för TextField-objekt och du kan använda CSS (Cascading Style Sheets) med egenskapen Textfield.styleSheet och klassen StyleSheet. Du kan tilldela HTML-formaterad text som kan innehålla inbäddade medier (filmklipp, SWF-filer, GIF-filer, PNG-filer och JPEG-filer) direkt till ett textfält.

Flash-textmotorn som är tillgänglig från och med Flash Player 10 och Adobe AIR 1.5, har lågnivåstöd för avancerad styrning av textmått, formatering och dubbelriktad text. Den har även bättre textflöde och utökat språkstöd. Du kan använda Flash Text Engine för att skapa och hantera textelement, men den är främst utformad som en grund för att skapa texthanteringskomponenter och kräver en högre nivå av programmeringskunskaper. Med Text Layout Framework, som omfattar en texthanteringskomponent baserad på Flash Text Engine, blir det enklare att använda avancerade funktioner i den nya textmotorn. Text Layout Framework är ett utökningsbart bibliotek helt skapat i ActionScript 3.0. Du kan använda den befintliga TLF-komponenten eller ramverket för att skapa egna textkomponenter.

Klassen StageText, som finns från och med AIR 3, har ett systemspecifikt textinmatningsfält. Eftersom det här fältet hämtas från enhetens operativsystem ser det ut precis som användaren förväntar sig. En StageText-instans är inget visningsobjekt. I stället för att lägga till det i visningslistan tilldelar du instansen en scen och ett visningsområde på den scenen, som kallas visningsport. StageText-instansen visas framför eventuella visningsobjekt.

Mer information om detta finns i:

- ”Använda klassen TextField” på sidan 373
- ”Använda Flash-textmotorn” på sidan 397
- ”Använda Text Layout Framework” på sidan 426
- Native text input with StageText

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer relaterade till texthantering:

CSS (Cascading Style Sheet) En standardsyntax för att ange stilar och formatering för innehåll som strukturerats i XML-format (eller HTML-format).

Enhetsteckensnitt Ett teckensnitt som är installerat på användarens maskin.

Dynamiskt textfält Ett textfält vars innehåll kan ändras med ActionScript men inte av användaren.

Inbäddat teckensnitt Ett teckensnitt vars konturdata lagras i programmets SWF-fil.

HTML-text Textinnehåll som anges i ett textfält med ActionScript och som inkluderar HTML-formateringstaggar tillsammans med textinnehållet.

Inmatningsfält Ett textfält vars innehåll kan ändras antingen av användaren eller med ActionScript.

Kerning En justering som gör mellanrummet mellan tecken mer proportionellt så att texten blir lättare att läsa.

Statiskt textfält Ett textfält som skapas i utvecklingsverktyget vars innehåll inte kan ändras när SWF-filen körs.

Textradsmått Storleksmått för olika delar av textinnehållet i textfältet, t.ex. textens baslinje, höjden på tecknens övre delar, storleken på nedstaplar (den del på vissa gemener som ligger under baslinjen) och så vidare.

Spärra/knip Justering av avståndet mellan grupper av tecken eller textblock som ökar eller minskar tätheten och gör texten lättare att läsa.

Kapitel 21: Använda klassen TextField

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda en instans av TextField-klassen när du vill visa text eller skapa ett textinmatningsfält på skärmen i Adobe® Flash® Player eller Adobe® AIR™. TextField-klassen är grunden för andra textbaserade komponenter, till exempel TextArea-komponenter eller TextInput-komponenter.

Textfältets innehåll kan anges i förväg i SWF-filen, hämtas från en textfil eller databas eller skrivas av den som använder ditt program. Texten kan återges som HTML-innehåll med inbäddade bilder i ett textfält. När du har skapat en instans av ett textfält kan du använda flash.text-klasser som TextFormat och StyleSheet för att ange textens utseende. Nästan alla klasser för att skapa, hantera och formatera text i ActionScript finns i flash.text-paketet.

Du kan formatera text genom att definiera formateringen i ett TextFormat-objekt och tilldela objektet till textfältet. Om textfältet innehåller HTML-text kan du använda ett StyleSheet-objekt för textfältet för att tilldela stilar till vissa delar av textfältets innehåll. TextFormat- eller StyleSheet-objektet innehåller egenskaper som definierar textens utseende, t.ex. färg, storlek och bredd. TextFormat-objektet tilldelar egenskaperna till allt innehåll i ett textfält eller ett textintervall. En mening kan t.ex. vara i röd fet stil och nästa mening kan vara i blå kursiv stil i ett textfält.

Förutom klasserna i flash.text-paketet kan du använda klassen flash.events.TextEvent när du vill svara på användaråtgärder som gäller texten.

Fler hjälpavsnitt

”Tilldela textformat” på sidan 380

”Visa HTML-text” på sidan 375

”Använda CSS (Cascading Style Sheets)” på sidan 380

Visa text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Utvecklingsverktyg som Adobe Flash Builder och Flash Professional har flera alternativ för textvisning, inklusive textrelaterade komponenter och textverktyg, men det enklaste sättet att programmässigt visa text är via ett textfält.

Typer av text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Typen av text i ett textfält karakteriseras av sin källa:

- Dynamisk text

  Dynamisk text inkluderar innehåll som läses in från en extern källa, t.ex. en textfil, en XML-fil eller en webbtjänst.

- Inmatningstext

Inmatningstext är text som användaren skriver eller dynamisk text som användaren kan redigera. Du kan ange en formatmall för att formatera inmatningstexten eller använda klassen flash.text.TextFormat om du vill tilldela egenskaper för det inmatade innehållet i textfältet. Mer information finns i ”Hämta textinmatningar” på sidan 378.

- Statisk text

  Statisk text skapas endast i Flash Professional. Du kan inte skapa en statisk textinstans med ActionScript 3.0, men du kan använda ActionScript-klasser som StaticText och TextSnapshot när du vill ändra en statisk textinstans. Mer information finns i ”Arbeta med statisk text” på sidan 386.

Ändra innehåll i textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan definiera dynamisk text genom att tilldela en sträng till egenskapen `flash.text.TextField.text`. Du kan tilldela en sträng direkt till egenskapen på följande sätt:

```
myTextField.text = "Hello World";
```

Du kan även tilldela `text`-egenskapen ett värde från en variabel som definierats i ditt skript, som i exemplet nedan:

```
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;
        }
    }
}
```

Alternativt kan du tilldela `text`-egenskapen ett värde från en fjärrvariabel. Det finns tre alternativ för att läsa in textvärden från fjärrkällor:

- Klasserna flash.net.URLLoader och flash.net.URLRequest läser in variabler för texten från en lokal plats eller fjärrplats.
- Attributet `FlashVars` är inbäddat i HTML-sidan som är värd för SWF-filen och det kan innehålla värden för textvariabler.
- Klassen flash.net.SharedObject hanterar upprepad lagring av värden. Mer information finns i ”Lagra lokala data” på sidan 703.

Visa HTML-text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen flash.text.TextField kan du använda egenskapen `htmlText` när du vill identifiera textsträngen som en sträng som innehåller HTML-taggar för formatering av innehåll. Du måste tilldela strängvärdet till egenskapen `htmlText` (inte `text`-egenskapen) för Flash Player eller AIR om du vill visa text som HTML, så som i exemplet nedan:

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player och AIR har stöd för en delmängd HTML-taggar och entiteter för egenskapen `htmlText`. Egenskapsbeskrivningen för `flash.text.TextField.htmlText` i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen innehåller detaljerad information om de HTML-taggar och entiteter som stöds.

När du anger innehåll med egenskapen `htmlText` kan du använda formatmallar eller taggen `textformat` när du vill formatera innehållet. Mer information finns i ”Formatera text” på sidan 380.

Använda bilder i textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En annan fördel med att visa innehållet som HTML-text är att du kan inkludera bilder i textfältet. Du kan skapa en referens till en lokal bild eller en fjärrbild med taggen `img` och visa den i det tillhörande textfältet.

Exemplet nedan skapar ett textfält kallat `myTextBox` som inkluderar en JPG-bild av ett öga som lagras i samma katalog som SWF-filen i texten som visas:

```
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;
        }
    }
}
```

Taggen `img` har stöd för JPEG-, GIF-, PNG- och SWF-filer.

Rulla text i ett textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I många fall kan din text vara längre än textfältet som visar texten. Eller du kan ha ett inmatningsfält där användaren kan skriva mer text än det som kan visas samtidigt. Du kan använda rullningsegenskaperna i klassen flash.text.TextField när du vill hantera långt innehåll, antingen lodrätt eller vågrätt.

Rullningsegenskaperna inkluderar `TextField.scrollV`, `TextField.scrollH` och `maxScrollV` samt `maxScrollH`. Använd egenskaperna när du vill svara på händelser, t.ex. ett musklick eller en tangenttryckning.

Exemplet nedan skapar ett textfält med en fast inställd storlek som innehåller mer text än fältet kan visa. Texten rullar lodrätt när användaren klickar på textfältet.

```
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++;
        }
    }
}
```

Markera och ändra text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan markera dynamisk text eller inmatningstext. Eftersom egenskaperna för textmarkering och metoderna för klassen TextField använder indexpositioner för att ange textintervallet som ska ändras, kan du programmässigt markera dynamisk text eller inmatningstext även om du inte känner till innehållet.

Obs! *Om du väljer markeringsalternativet för ett statiskt textfält i Flash Professional kommer textfältet som exporteras och placeras i visningslistan att vara ett vanligt dynamiskt textfält.*

Markera text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskapen `flash.text.TextField.selectable` är `true` som standard och du kan programmässigt markera texten med metoden `setSelection()`.

Du kan till exempel ange att en viss text i ett textfält ska markeras när användaren klickar på textfältet:

```
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);
}
```

Om du vill att text i ett textfält ska markeras när texten visas kan du skapa en händelsehanterarfunktion som anropas när textfältet läggs till i visningslistan.

Hämta användarmarkerad text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

TextField-egenskaperna `selectionBeginIndex` och `selectionEndIndex`, som är ”skrivskyddade” så att de inte kan ställas in att programmässigt markera text, kan användas för att hämta det som användaren har markerat. Textinmatningsfält kan även använda egenskapen `caretIndex`.

Koden nedan spårar t.ex. indexvärdena för användarmarkerad text:

```
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);
}
```

Du kan använda ett urval av egenskaper för TextFormat-objekt när du vill ändra den markerade textens utseende. Mer information om att använda en uppsättning TextFormat-egenskaper för markerad text finns i ”Formatera textintervall i ett textfält” på sidan 383.

Hämta textinmatningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som standard är egenskapen `type` för ett textfält inställt på `dynamic`. Om du ställer in egenskapen `type` på `input` med klassen TextFieldType, kan du samla in användarindata och spara värdena för användning i andra delar av programmet. Inmatningsfält är praktiska för formulär och andra program där användaren ska definiera ett textvärde för användning på andra ställen i programmet.

Koden nedan skapar t.ex. ett inmatningsfält kallat `myTextBox`. En `textInput`-händelse aktiveras när användaren skriver text i fältet. En händelsehanterare som kallas `textInputCapture` hämtar den skrivna textsträngen och tilldelar den en variabel. Flash Player eller AIR visar den nya texten i ett annat textfält kallat `myOutputBox`.

```
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;
        }

    }
}
```

Begränsa textinmatningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom inmatningsfält ofta används för formulär och dialogrutor i program, kanske du vill begränsa de typer av tecken som användaren kan skriva i ett textfält eller dölja texten för t.ex. ett lösenord. Klassen flash.text.TextField har egenskaperna `displayAsPassword` och `restrict` som du kan använda när du vill styra användarens inmatningar.

Egenskapen `displayAsPassword` döljer texten (en serie asterisker visas) allt eftersom användaren skriver texten. Kommandona Klipp ut och Kopiera samt motsvarande kortkommandon fungerar inte när `displayAsPassword` är inställt på `true`. Du tilldelar egenskapen `displayAsPassword` på samma sätt som andra egenskaper, t.ex. bakgrund och färg, så som exemplet nedan visar:

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

Egenskapen `restrict` är lite mer komplicerad eftersom du måste ange vilka tecken som användaren får skriva i ett inmatningsfält. Du kan tillåta specifika bokstäver och siffror eller intervall av bokstäver, siffror och tecken. Koden nedan gör att användaren bara kan skriva versaler (inte siffror eller specialtecken) i textfältet:

```
myTextBox.restrict = "A-Z";
```

I ActionScript 3.0 används bindestreck för att definiera intervall och cirkumflex för att definiera uteslutna tecken. Mer information om hur du definierar vad som är begränsat i ett textfält finns i avsnittet om egenskapen `flash.text.TextField.restrict` i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Obs! *Om du använder egenskapen `flash.text.TextField.restrict` kommer bokstäver som inte är tillåtna automatiskt att konverteras till tillåtna. Om du använder egenskapen `fl.text.TLFTextField.restrict` (dvs. om du använder ett TLF-textfält) kommer de otillåtna tecknen att ignoreras.*

Formatera text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns flera alternativ för att programmässigt formatera den visade texten. Du kan ange egenskaper direkt i instansen TextField — till exempel egenskaperna `TextFIeld.thickness`, `TextField.textColor` och `TextField.textHeight` .Du kan även ange innehåll i textfältet med egenskapen `htmlText` och använda HTML-taggar som stöds, t.ex. `b`, `i` och `u`. Du kan också använda TextFormat-objekt i textfält som innehåller oformaterad text och StyleSheet-objekt i textfält som innehåller egenskapen `htmlText`. Objekten TextFormat och StyleSheet ger dig störst kontroll och enhetlighet för textobjekten i hela programmet. Du kan definiera ett TextFormat- eller StyleSheet-objekt och använda det på flera eller alla textfält i programmet.

Tilldela textformat

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda klassen TextFormat när du vill ange ett antal egenskaper för visning av text och sedan tillämpa dem på allt innehåll i ett TextField-objekt eller ett textintervall.

I exemplet nedan används ett TextFormat-objekt för ett helt TextField-objekt och ett annat TextFormat-objekt används för ett intervall av text i samma TextField-objekt:

```
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);
```

Metoden `TextField.setTextFormat()` påverkar bara text som redan visas i textfältet. Om innehållet i textfältet ändras, kanske programmet måste anropa metoden `TextField.setTextFormat()` igen för att göra om formateringen. Du kan även ange att textfältsegenskapen `defaultTextFormat` ska specificera formateringen som används för text som användaren skriver.

Använda CSS (Cascading Style Sheets)

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Textfält kan innehålla oformaterad text eller HTML-formaterad text. Oformaterad text lagras i egenskapen `text` för instansen och HTML-text lagras i egenskapen `htmlText`.

Du kan använda CSS formatdeklarationer när du vill definiera textformat som kan användas på flera olika textfält. CSS-formatdeklarationer kan skapas i programkoden eller läsas in vid körningstillfället från en extern CSS-fil.

Klassen flash.text.StyleSheet hanterar CSS-format. Klassen StyleSheet identifierar en begränsad uppsättning av CSS-egenskaper. En utförlig lista över de formategenskaper som klassen StyleSheet har stöd för finns i avsnittet om flash.textStylesheet i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Som exemplet nedan visar, kan du skapa CSS i koden och tillämpa formaten för HTML-text genom att använda ett StyleSheet-objekt:

```
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);
```

I exempelkoden skapas ett enkelt objekt som lagrar en uppsättning formatdeklarationsegenskaper efter det att du skapat ett StyleSheet-objekt. Sedan anropas metoden `StyleSheet.setStyle()` som lägger till den nya formateringen i formatmallen med namnet ".darkred". Formateringen i formatmallen används genom att objektet StyleSheet tilldelas TextField-egenskapen `styleSheet`.

För att CSS-format ska aktiveras bör formatmallen tillämpas på TextField-objektet innan egenskapen `htmlText` anges.

Ett textfält med en formatmall kan inte redigeras. Om du har ett textinmatningsfält och tilldelar det en formatmall, visar textfältet formatmallens egenskaper, men användare kan inte skriva ny text i textfältet. Du kan inte heller använda följande ActionScript API:er på ett textfält som tilldelats en formatmall:

- Metoden `TextField.replaceText()`
- Metoden `TextField.replaceSelectedText()`
- Egenskapen `TextField.defaultTextFormat`
- Metoden `TextField.setTextFormat()`

Om ett textfält har en formatmall tilldelad och egenskapen `TextField.styleSheet` ställs in på `null`, kommer innehållet i egenskaperna `TextField.text` och `TextField.htmlText` att lägga till taggar och attribut i innehållet för att inkludera formateringen från den tilldelade formatmallen. För att bevara den ursprungliga `htmlText`-egenskapen, ska du spara den som en variabel innan formatmallen ställs in på `null`.

Läsa in en extern CSS-fil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

CSS-metoden för formatering är effektivare om du läser in CSS-information från en extern fil vid körningstillfället. När CSS-data är externa i förhållande till programmet kan du ändra den visuella stilen för text i programmet utan att ändra ActionScript 3.0 källkoden. När programmet har distribuerats kan du ändra en extern CSS-fil om du vill förändra programmets utseende, utan att behöva omdistribuera programmets SWF-fil.

Metoden `StyleSheet.parseCSS()` konverterar en sträng som innehåller CSS-data till formatdeklarationer i StyleSheet-objektet. Följande exempel visar hur du läser en extern CSS-fil och tillämpar filens formatdeklarationer på ett TextField-objekt.

Här är innehållet i CSS-filen som ska läsas in och den kallas example.css:

```
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;
}
```

Sedan följer ActionScript-koden för en klass som läser in example.css-filen och tillämpar formateringen på textfältsinnehåll:

```
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;
        }
    }
}
```

När CSS-data har lästs in, körs metoden `onCSSFileLoaded()` och anropar metoden `StyleSheet.parseCSS()` för överföring av formatdeklarationerna till StyleSheet-objektet.

Formatera textintervall i ett textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En användbar metod för klassen flash.text.TextField är metoden `setTextFormat()`. Med `setTextFormat()` kan du tilldela specifika egenskaper till innehållet i en del av ett textfält som svarar på användarinmatningar, t.ex. formulär som påminner användaren om att vissa inmatningar krävs, eller för att ändra betoningen av ett underavsnitt i ett textstycke när användaren markerar delar av texten.

I exemplet nedan används `TextField.setTextFormat()` för ett intervall av tecken för att ändra utseendet på en del av innehållet i `myTextField` när användaren klickar på textfältet:

```
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);
}
```

Avancerad textåtergivning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 finns ett antal klasser i paketet flash.text som styr egenskaperna för den visade texten, inklusive inbäddade teckensnitt, kantutjämningsinställningar, styrning av alfakanaler och andra specifika inställningar. Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen innehåller detaljerade beskrivningar av dessa klasser och egenskaper, inklusive klasserna CSMSettings, Font och TextRenderer.

Använda inbäddade teckensnitt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du anger ett visst teckensnitt för ett textfält i ditt program, söker Flash Player eller AIR efter ett enhetsteckensnitt (ett teckensnitt på användarens dator) med samma namn. Om teckensnittet inte finns i systemet eller om användaren har en annan version av teckensnittet, kan texten som visas se annorlunda ut än du tänkt dig. Texten visas som standard med teckensnittet Times Roman.

Du kan bädda in teckensnittet i programmets SWF-fil om du vill vara säker på att användaren ska se exakt rätt teckensnitt. Inbäddade teckensnitt har flera fördelar:

- Inbäddade teckensnitt är kantutjämnade vilket gör att kanterna ser jämnare ut, speciellt när texten är stor.
- Du kan rotera text som använder inbäddade teckensnitt.
- Inbäddade teckensnitt kan vara genomskinliga eller halvgenomskinliga.
- Du kan använda CSS-formatet `kerning` med inbäddade teckensnitt.

Den största nackdelen med inbäddade teckensnitt är att de ökar filstorleken eller hämtningsstorleken på ditt program.

Den exakta metoden för att bädda in en teckensnittsfil i programmets SWF-fil varierar beroende på utvecklingsmiljön.

Du kan kontrollera att ett textfält använder rätt inbäddat teckensnitt när du har bäddat in det:

- Ställ in egenskapen `embedFonts` för textfältet på `true`.
- Skapa ett TextFormat-objekt, ställ in egenskapen `fontFamily` på namnet på det inbäddade teckensnittet och tillämpa TextFormat-objektet på textfältet. När du specificerar ett inbäddat teckensnitt bör egenskapen `fontFamily` bara innehålla ett namn, du kan inte använda en kommaavgränsad lista med flera teckensnittsnamn.
- Om du använder CSS-format för att ange teckensnitt för textfält eller komponenter, ska du ställa in CSS-egenskapen `font-family` på det inbäddade teckensnittets namn. Egenskapen `font-family` får bara innehålla ett namn, inte en lista över namn, när du vill ange ett inbäddat teckensnitt.

Bädda in ett teckensnitt i Flash

Med Flash Professional kan du bädda in nästan vilket teckensnitt som helst som är installerat i systemet, inklusive TrueType-teckensnitt och Type 1 Postscript-teckensnitt.

Du kan bädda in teckensnitt i ett program på många sätt, bland annat:

- Ange teckensnittet och stilegenskaperna för ett textfält på scenen och klicka på kryssrutan Bädda in teckensnitt
- Skapa och ange en referens för en teckensnittssymbol
- Skapa och använda ett delat bibliotek som innehåller inbäddade teckensnittssymboler vid körning

Mer information om att bädda in teckensnitt i program finns i ”Inbäddade teckensnitt för dynamiska textfält och inmatningsfält” i *Använda Flash*.

Bädda in ett teckensnitt i Flex

Du kan bädda in teckensnitt i ett Flex-program på många sätt, bland annat:

- Använda metadatataggen `[Embed]` i ett skript
- Med formatdeklarationen `@font-face`
- Upprätta en klass för teckensnittet och bädda in det med hjälp av taggen `[Embed]`.

Du kan bara bädda in TrueType-teckensnitt direkt i ett Flex-program. Teckensnitt med andra format, t.ex. Type 1 Postscript-teckensnitt, kan först bäddas in i en SWF-fil med Flash Professional och sedan kan SWF-filen användas i Flex-programmet. Mer information om att använda inbäddade teckensnitt från SWF-filer i Flex finns i avsnittet om att bädda in teckensnitt från SWF-filer i *Använda Flex 4*.

Fler hjälpavsnitt

Bädda in teckensnitt för enhetlig text

Peter deHaan: Embedding fonts (Bädda in teckensnitt)

Divillysausages.com: AS3 Font embedding masterclass (Masterclass i AS3-teckensnittsinbäddning)

Reglera skärpa, tjocklek och kantutjämning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som standard fastställer Flash Player eller AIR inställningarna för textvisning, t.ex. skärpa, tjocklek och kantutjämning, allt eftersom textens storlek eller färg ändras eller texten visas på olika bakgrunder. I vissa fall, t.ex. när du har mycket liten eller mycket stor text eller text på olika unika bakgrunder, kanske du vill ha kontroll över inställningarna. Du kan åsidosätta Flash Player- eller AIR-inställningarna genom att använda klassen `flash.text.TextRenderer` och tillhörande klasser, t.ex. CSMSettings. Dessa klasser ger dig fullständig kontroll över den inbäddade textens visningskvalitet. Mer information om inbäddade teckensnitt finns i ”Använda inbäddade teckensnitt” på sidan 384.

Obs! *Egenskapen `flash.text.TextField`.antiAliasType måste ha värdet `AntiAliasType.ADVANCED` för att du ska kunna ställa in skärpa, tjocklek och egenskapen gridFitType eller om du vill använda metoden `TextRenderer.setAdvancedAntiAliasingTable()`.*

Exemplet nedan använder anpassade CSM-egenskaper (continuous stroke modulation) och formatering för text som visas med ett inbäddat teckensnitt kallat `myFont`. Flash Player eller Adobe AIR tillämpar de anpassade inställningarna när användaren klickar på texten som visas:

```
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);
}
```

Arbeta med statisk text

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Statisk text skapas endast i Flash Professional. Du kan inte programmässigt instansiera statisk text med ActionScript. Statisk text kan användas när texten är kort och texten inte är avsedd att ändras (som dynamisk text kan). Du kan tänka dig statisk text ungefär som ett grafikelement, till exempel en cirkel eller fyrkant, som ritas på scenen i Flash Professional. Trots att statisk text är mer begränsad än dynamisk text, kan du läsa egenskapsvärden för statisk text med klassen StaticText i ActionScript 3.0. Du kan även använda klassen TextSnapshot när du vill läsa värden från den statiska texten.

Öppna statiska textfält med klassen StaticText

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I vanliga fall använder du klassen flash.text.StaticText på åtgärdspanelen i Flash Professional när du ska arbeta med en statisk textinstans på scenen. Du kan även arbeta i ActionScript-filer som interagerar med en SWF-fil som innehåller statisk text. Oavsett vilket du använder, kan du inte instansiera en statisk textinstans programmässigt. Statisk text skapas i Flash Professional.

Gå igenom objekten i visningslistan och tilldela en variabel när du vill skapa en referens för ett befintligt statiskt textfält. Till exempel:

```
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);
}
}
```

När du har en referens till ett statiskt textfält kan du använda egenskaperna för det fältet i ActionScript 3.0. Koden nedan är kopplad till en bildruta på tidslinjen och förutsätter att en variabel kallad `myFieldLabel` har tilldelats en statisk textreferens. Ett dynamiskt textfält kallat `myField` är positionerat i förhållande till `x`- och `y`-värdena för `myFieldLabel` och visar värdet för `myFieldLabel` igen.

```
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
```

Använda klassen TextSnapshot

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du programmässigt vill arbeta med en befintlig statisk textinstans, kan du använda klassen flash.text.TextSnapshot och arbeta med egenskapen `textSnapshot` för en flash.display.DisplayObjectContainer. Du skapar med andra ord en TextSnapshot-instans av egenskapen `DisplayObjectContainer.textSnapshot`. Du kan sedan tillämpa metoder på instansen när du vill hämta värden eller markera delar av den statiska texten.

Du kan t.ex. placera ett statiskt textfält som innehåller texten ”TextSnapshot Example” på scenen. Lägg till följande ActionScript till bildruta 1 på tidslinjen:

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

Klassen TextSnapshot är praktisk när du vill hämta text från statiska textfält i en inläst SWF-fil och använda texten som ett värde i en annan del av ett program.

Exempel på textfält: Textformatering i tidningsstil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Exemplet News Layout formaterar text så att den ser ut som en artikel i en tryckt tidning. Den inmatade texten kan innehålla en rubrik, en underrubrik och artikelns brödtext. I exemplet News Layout formateras rubriken och underrubriken så att visningsområdets hela bredd utnyttjas när visningsområdets bredd och höjd anges. Brödtexten fördelas på två eller flera kolumner.

Exemplet visar följande ActionScript-programmeringstekniker:

- Utöka klassen TextField
- Läsa in och använda en extern CSS-fil
- Konvertera CSS-format till TextFormat-objekt
- Hämta information om den visade textens storlek med klassen TextLineMetrics

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. News Layout-programfilerna finns i mappen Samples/NewsLayout. Programmet består av följande filer:

Fil	Beskrivning
NewsLayout.mxml eller NewsLayout.fla	Användargränssnittet för programmet för Flex (MXML) eller Flash (FLA).
com/example/programmingas3/newslayout/StoryLayoutComponent.as	En Flex UIComponent-klass som placerar StoryLayout-instansen.
com/example/programmingas3/newslayout/StoryLayout.as	ActionScript-huvudklassen som arrangerar alla komponenter i en tidningsartikel för visning.
com/example/programmingas3/newslayout/FormattedTextField.as	En underklass av klassen TextField som hanterar sitt eget TextFormat-objekt.
com/example/programmingas3/newslayout/HeadlineTextField.as	En underklass av klassen FormattedTextField som justerar teckensnittsstorlekarna efter önskad bredd.
com/example/programmingas3/newslayout/MultiColumnTextField.as	En ActionScript-klass som fördelar text på två eller fler kolumner.
story.css	En CSS-fil som definierar layoutens textstilar.

Läsa den externa CSS-filen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

News Layout-programmet börjar med att läsa artikeltexten från en lokal XML-fil. Sedan läser det en extern CSS-fil som innehåller formateringsinformation för rubriken, underrubriken och brödtexten.

I CSS-filen definieras tre stilar, ett vanligt styckeformat för artikeln samt stilarna h1 och h2 för rubriken respektive underrubriken.

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

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;
}
```

Tekniken som används för att läsa den externa CSS-filen är densamma som tekniken som beskrivs i ”Läsa in en extern CSS-fil” på sidan 381. Programmet kör metoden `onCSSFileLoaded()` när CSS-filen har lästs in, så som visas nedan.

```
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;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

Metoden `onCSSFileLoaded()` skapar ett StyleSheet-objekt och tolkar data från CSS-filen. Brödtexten i artikeln visas i ett MultiColumnTextField-objekt som kan använda ett StyleSheet-objekt direkt. Rubrikfälten använder klassen HeadlineTextField som använder ett TextFormat-objekt för formateringen.

Metoden `onCSSFileLoaded()` anropar metoden `getTextStyle()` två gånger för att konvertera en CSS-formatdeklaration till ett TextFormat-objekt som kan användas för de två HeadlineTextField-objekten.

```
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.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

Egenskapsnamnen och egenskapsvärdenas betydelse skiljer sig mellan CSS-formatdeklarationer och TextFormat-objekt. Metoden `getTextStyle()` översätter CSS-egenskapsvärden till värden som förväntas av TextFormat-objektet.

Arrangera artikelelementen på sidan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen StoryLayout formaterar och arrangerar rubriken, underrubriken och brödtextfälten i en tidningslayout. Metoden `displayText()` skapar och placerar de olika fälten inledningsvis.

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

Varje fält placeras under det föregående fältet genom att dess `y`-egenskap ställs in på samma `y`-egenskap som det föregående fältet plus dess höjd. Denna dynamiska beräkning av placeringen krävs eftersom objekten HeadlineTextField och MultiColumnTextField kan ändra höjd beroende på innehållet.

Ändra teckensnittsstorleken så att den passar fältstorleken

Flash Player 9 och senare, Adobe AIR 1.0 och senare

HeadlineTextField ändrar teckensnittets storlek så att texten passar den bredd i pixlar och det maximala antalet rader som angetts för fältet. Om texten är kort blir teckenstorleken stor, vilket skapar en rubrik i tabloidstil. Om texten är lång blir teckensnittsstorleken mindre.

Metoden `HeadlineTextField.fitText()` som visas nedan ändrar teckensnittsstorleken:

```
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)
    {
        // the point size is too small
        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)
    {
        // set it back to the last size
        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;
    }
}
```

Metoden `HeadlineTextField.fitText()` använder en enkel rekursiv teknik när teckensnittets storlek ändras. Först uppskattas det genomsnittliga antalet pixlar per tecken i texten och sedan beräknas en inledande punktstorlek baserat på detta. Därefter ändras teckensnittsstorleken och metoden kontrollerar om texten har automatisk radbrytning som skapar mer än det maximala antalet textrader. Metoden `shrinkText()` anropas för att minska teckensnittsstorleken och försöka igen om det finns för många rader. Metoden `growText()` anropas för att öka teckensnittsstorleken och försöka igen om det inte finns för många rader. Processen avbryts när för många rader skapas om teckensnittsstorleken ökas med en punkt.

Fördela text över flera kolumner

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen MultiColumnTextField fördelar text över flera TextField-objekt som sedan ordnas som tidningskolumner.

Konstruktorn `MultiColumnTextField()` skapar först en array av TextField-objekt, ett för varje kolumn, så som visas här:

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

Varje TextField-objekt läggs till i arrayen och i visningslistan med metoden `addChild()`.

När egenskapen StoryLayout `text` eller `styleSheet`ändras, anropas metoden `layoutColumns()` för att visa texten på nytt. Metoden `layoutColumns()` anropar metoden `getOptimalHeight()` som beräknar pixelhöjden som krävs för att få plats med all texten inom den angivna bredden.

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

Metoden `getOptimalHeight()` beräknar först bredden på varje kolumn. Sedan ställs bredden och egenskapen `htmlText` in för det första TextField-objektet i arrayen. Metoden `getOptimalHeight()` använder det första TextField-objektet för att identifiera det totala antalet rader med automatisk radbrytning i texten och identifierar med hjälp av detta hur många rader varje kolumn ska ha. Sedan anropas metoden `TextField.getLineMetrics()` som hämtar ett TextLineMetrics-objekt som innehåller information om storleken på texten på den första raden. Egenskapen `TextLineMetrics.height` representerar den totala höjden för en textrad i pixlar, inklusive nedstapel, uppstapel och radavstånd. Den optimala höjden för MultiColumnTextField-objektet är radhöjden multiplicerad med antalet rader per kolumn plus 4 för kanten på två pixlar längst upp och längst ned på ett TextField-objekt.

Det här är koden för den fullständiga `layoutColumns()`-metoden:

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;

        field.height = this.preferredHeight;
        field.text = remainder;

        field.setTextFormat(this.format);
```

```
var lineLen:int;
if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
        }
    }

field.x = i * (colWidth + gutter);
field.y = 0;

remainder = "";
fieldText = "";

var linesRemaining:int = field.numLines;
var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

for (var j:int = 0; j < linesRemaining; j++)
{
    if (j < linesVisible)
    {
        fieldText += field.getLineText(j);
    }
    else
    {
        remainder +=field.getLineText(j);
    }
}

field.text = fieldText;

field.setTextFormat(this.format);

if (indent > 0 && !lastLineEndedPara)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
    }
}

var lastLine:String = field.getLineText(field.numLines - 1);
var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);
```

```
        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

När egenskapen `preferredHeight` har ställts in genom att metoden `getOptimalHeight()` anropas, går metoden `layoutColumns()` igenom TextField-objekten och ställer in höjden för varje objekt på värdet `preferredHeight`. Metoden `layoutColumns()` distribuerar sedan tillräckligt med textrader till varje fält så att de enskilda fälten inte rullar och texten i nästa fält börjar där texten i det föregående fältet tar slut. Om textjustering har ställts in på ”justera” anropas metoden `justifyLastLine()` för att justera den sista textraden i ett fält. Annars behandlas den sista raden som ett styckeslut och justeras inte.

Kapitel 22: Använda Flash-textmotorn

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Flash-textmotorn Adobe® Flash® Text Engine (FTE) som är tillgänglig från och med Flash Player 10 och Adobe® AIR™1.5, har lågnivåstöd för avancerad styrning av textmått, formatering och dubbelriktad text. Den ger bättre textflöde och har utökat språkstöd. Även om FTE kan användas för att skapa och hantera enkla textelement så är FTE främst en grund för utvecklare som skapar texthanteringskomponenter. Det innebär att en högre nivå av programmeringskunskaper krävs för Flash-textmotorn. Mer information om hur du visar enkla textelement finns i ”Använda klassen TextField” på sidan 373.

Text Layout Framework, som innehåller en komponent för texthantering baserat på FTE, är ett enklare sätt att använda dess avancerade funktioner. Text Layout Framework är ett utökningsbart bibliotek helt skapat i ActionScript 3.0. Du kan använda den befintliga TLF-komponenten eller ramverket för att skapa egna textkomponenter. Mer information finns i ”Använda Text Layout Framework” på sidan 426.

Fler hjälpavsnitt

flash.text.engine-paket

Skapa och visa text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med klasserna i Flash-textmotorn kan du skapa, formatera och styra text. Följande klasser är de grundläggande byggstenarna när du skapar och visar text med Flash-textmotorn:

- TextElement/GraphicElement/GroupElement - lagrar innehållet i en TextBlock-instans
- ElementFormat - anger formateringsattribut för innehållet i en TextBlock-instans
- TextBlock - används när du skapar ett textstycke
- TextLine - en textrad som skapats av ett TextBlock

Visa texten genom att skapa ett TextElement-objekt från en med ett ElementFormat-objekt för att ange formateringsegenskaperna. Tilldela TextElement till egenskapen `content` för ett TextBlock-objekt. Skapa textraderna som ska visas genom att anropa metoden `TextBlock.createTextLine()`. Metoden `createTextLine()` returnerar ett TextLine-objekt som innehåller så mycket av strängen som ryms i den angivna bredden. Anropa metoden upprepade gånger tills hela strängen har formaterats till olika rader. När alla rader har skapats tilldelas textLineCreationResult-egenskapen för TextBlock-objektet värdet: `TextLineCreationResult.COMPLETE`. Om du vill visa raderna lägger du till dem i visningslistan (med respektive `x`- och `y`-positionsvärden).

I koden nedan används dessa FTE-klasser för att visa texten ”Hello World! This is Flash Text Engine!” med standardvärden för format och teckensnitt. I det här enkla exemplet har endast en textrad skapats.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

Parametrarna för `createTextLine()` anger raden där den nya raden ska börja och radens bredd i pixlar. Raden från vilken nästa rad börjar är vanligtvis den föregående raden, men för den första raden är värdet `null`.

Lägga till GraphicElement- och GroupElement-objekt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan tilldela ett GraphicElement-objekt till ett TextBlock-objekt när du vill visa en bild eller ett grafikelement. Skapa bara en instans av GraphicElement-klassen av en grafikbild eller annan bild och tilldela instansen till egenskapen `TextBlock.content`. Skapa textraden genom att anropa `TextBlock.createTextline()` som vanligt. I exemplet nedan skapas två textrader, en med ett GraphicElement-objekt och en med ett TextElement-objekt.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();
```

```
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

Du kan skapa ett GroupElement-objekt om du vill skapa en grupp av TextElement-, GraphicElement- och andra GroupElement-objekt. Ett GroupElement kan tilldelas till egenskapen `content` för ett TextBlock-objekt. Parametern för `GroupElement()`-konstruktorn är en vektor som pekar till text-, grafik- och gruppelementen som ingår i gruppen. I exemplet nedan grupperas två grafikelement och ett textelement och de tilldelas som en enhet till ett textblock.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
```

```
            triangle2.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1,
triangle1.width, triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2,
triangle2.width, triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

Ersätta text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan ersätta text i en TextBlock-instans genom att anropa `TextElement.replaceText()` för att ersätta text i det textelement som du tilldelade till egenskapen `TextBlock.content`.

I exemplet nedan används `replaceText()` för att först infoga text i början av raden, sedan lägga till text i slutet av raden och slutligen ersätta text i mitten av raden.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

Metoden `replaceText()` ersätter texten som anges med parametrarna `beginIndex` och `endIndex` med texten som anges i parametern `newText`. Om värdena för parametrarna `beginIndex` och `endIndex` är lika, infogar `replaceText()` den angivna texten på den platsen. Annars ersätts tecknen som anges med `beginIndex` och `endIndex` av den nya texten.

Hantera händelser i FTE

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan lägga till händelseavlyssnare till en TextLine-instans precis som du kan för andra visningsobjekt. Du kan t.ex. identifiera när en användare rullar musen över en textrad eller klickar på raden. Båda händelserna identifieras i exemplet nedan. När du rullar musen över raden ändras markören till en knappmarkör och när du klickar på raden ändrar den färg.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
```

```
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

Spegla händelser

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan spegla händelser i ett textblock eller en del av ett textblock till en händelsedispatcher. Skapa först en EventDispatcher-instans och tilldela den sedan till egenskapen `eventMirror` för en TextElement-instans. Om textblocket består av ett textelement, speglar textmotorn händelser för hela textblocket. Om textblocket består av flera textelement, speglar textmotorn bara händelser för de TextElement-instanser som har egenskapen `eventMirror`. Texten i exemplet nedan består av tre element: ordet ”Click”, ordet ”here” och strängen ”to see me in italic”. Exemplet tilldelar en händelsedispatcher till det andra textelementet, ordet ”here” och lägger till en händelseavlyssnare, metoden `clickHandler()`. Metoden `clickHandler()` ändrar texten till kursiv text. Den ersätter även innehållet i det tredje textelementet så att det står ”Click here to see me in normal font!”.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();
```

```
            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

Funktionerna `mouseOverHandler()` och `mouseOutHandler()` ändrar markören till en knappmarkör när markören befinner sig på ordet ”here” och ändrar den tillbaks till en pil när den inte är det.

Formatera text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Ett `TextBlock`-objekt används när du vill skapa rader av text. Innehållet i ett `TextBlock`-objekt tilldelas via objektet `TextElement`. Ett `ElementFormat`-objekt hanterar textens formatering. Klassen ElementFormat definierar egenskaper som baslinjejustering, kerning, spårning, textrotation och teckensnittsstorlek, färg samt skiftläge. Den inkluderar även en `FontDescription` som behandlas utförligt i ”Arbeta med teckensnitt” på sidan 409.

Använda objektet ElementFormat

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Konstruktorn för objektet `ElementFormat` hanterar vilka som helst i en lång lista över alternativa parametrar, inklusive en `FontDescription`. Du kan även ange egenskaperna utanför konstruktorn. Exemplet nedan visar förhållandet mellan de olika objekten när en enkel textrad definieras och visas:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Teckenfärg och genomskinlighet (alfa)

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `color` i objektet `ElementFormat` anger teckenfärgen. Värdet är ett heltal som representerar färgens RGB-komponenter, t.ex. 0xFF0000 för rött och 0x00FF00 för grönt. Standardvärdet är svart (0x000000).

Egenskapen `alpha` ställer in alfagenomskinlighetsvärdet för ett element (både `TextElement` och `GraphicElement`). Värdena kan gå från 0 (helt genomskinlig) till 1 (helt ogenomskinlig vilket är standardvärdet). Element med ett `alfavärde` på 0 är osynliga men fortfarande aktiva. Värdet multipliceras med eventuella ärvda alfavärden vilket gör elementet mer genomskinligt.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

Baslinjejustering och förskjutning

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Teckensnittet och storleken på den största texten på en rad avgör den dominerande baslinjen. Du kan åsidosätta dessa värden genom att ställa in `TextBlock.baselineFontDescription` och `TextBlock.baselineFontSize`. Du kan justera den dominanta baslinjen med en av flera baslinjer i texten. Dessa baslinjer inkluderar uppstapeln och nedstapeln eller den ideografiska överdelen, mittdelen eller underdelen.

***A.** Uppstapel **B.** Baslinje **C.** Nedstapel **D.** x-höjd*

Tre egenskaper avgör egenskaperna för baslinjen och justeringen i objektet `ElementFormat`. Egenskapen `alignmentBaseline` ställer in huvudbaslinjen för ett `TextElement` eller `GraphicElement`. Baslinjen är ”fästlinjen” för elementet och det är efter det läget som den dominerande baslinjen för all text justeras.

Egenskapen `dominantBaseline` anger vilka av elementets olika baslinjer som ska användas och som anger den vertikala positionen för elementet på den raden. Standardvärdet är `TextBaseline.ROMAN`, men du kan även ange att baslinjen `IDEOGRAPHIC_TOP` eller `IDEOGRAPHIC_BOTTOM` ska vara dominerande.

Egenskapen `baselineShift` flyttar baslinjen med ett angivet antal pixlar längs y-axeln. Med normal (ej roterad) text flyttar ett positivt värde baslinjen nedåt och ett negativt värde flyttar den uppåt.

Typografiskt skiftläge

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `TypographicCase` i `ElementFormat` anger textens skiftläge, t.ex. versaler, gemener eller kapitäler.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

Rotera text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan rotera ett textblock eller glyferna i ett textsegment i steg om 90 grader. Klassen TextRotation definierar följande konstanter för inställning av både textblock och glyfernas rotering:

Konstant	Värde	Beskrivning
AUTO	”auto”	Anger 90 graders rotation moturs. Används i vanliga fall med vertikal asiatisk text för att enbart rotera glyfer som kräver rotering.
ROTATE_0	”rotate_0”	Anger ingen rotation.
ROTATE_180	”rotate_180”	Anger 180 graders rotering.
ROTATE_270	”rotate_270”	Anger 270 graders rotering.
ROTATE_90	”rotate_90”	Anger 90 graders rotation medurs.

Ställ in egenskapen `TextBlock.lineRotation` innan du anropar metoden `TextBlock.createTextLine()` för att skapa textraden om du vill rotera textrader i ett textblock.

När du vill rotera glyfer i ett textblock eller -segment ska du ställa in egenskapen `ElementFormat.textRotation` på det antal grader som du vill rotera glyferna. En glyf är den form som skapar ett tecken eller en del av ett tecken som består av flera glyfer. Bokstaven ”a” och pricken över ett ”i” är exempel på glyfer.

Roterade glyfer används för vissa asiatiska språk när du vill rotera linjer vertikalt men inte tecknen på raderna. Mer information om att rotera asiatisk text finns i ”Justera östasiatisk text” på sidan 413.

Nedan finns ett exempel hur du roterar både ett textblock och glyferna i det så som du gör för asiatisk text. I exemplet används även ett japanskt teckensnitt:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Låsa och klona ElementFormat

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När ett `ElementFormat`-objekt tilldelas en typ av `ContentElement`, ställs egenskapen `locked` automatiskt in på `true`. Om du försöker ändra ett låst `ElementFormat`-objekt, inträffar felet `IllegalOperationError`. Det bästa sättet är att helt definiera ett sådant objekt innan det tilldelas till en `TextElement`-instans.

Om du vill ändra en befintlig `ElementFormat`-instans kontrollerar du först egenskapen `locked`. Om egenskapen är `true` kan du skapa en olåst kopia av objektet med metoden `clone()`. Du kan ändra egenskaperna för det olåsta objektet och sedan tilldela det till `TextElement`-instansen. Eventuella nya rader som skapas kommer att ha den nya formateringen. Rader som skapats tidigare för samma objekt med den gamla formateringen ändras inte.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Arbeta med teckensnitt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Objektet `FontDescription` används tillsammans med `ElementFormat` för att identifiera ett typsnitt och definiera vissa av dess egenskaper. Egenskaperna inkluderar teckensnittsnamn, bredd, läge, återgivning och var teckensnittet finns (på enheten eller inbäddat).

Obs! *FTE har inte stöd för Type 1-teckensnitt eller bitmappade teckensnitt som Type 3, ATC, sfnt-wrapped CID och Naked CID.*

Definiera teckensnittsegenskaper (objektet FontDescription)

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `fontName` i `FontDescription`-objektet kan vara ett namn eller en kommaavgränsad lista med namn. I en lista som t.ex. ”Arial, Helvetica, _sans”, söker textmotorn först efter ”Arial”, sedan ”Helvetica” och till slut efter ”_sans” om programmet inte kan hitta de två första teckensnitten. Uppsättningen med teckensnittsnamn inkluderar tre namn på generiska enhetsteckensnitt: ”_sans”, ”_serif” och ”_typewriter”. De hänvisar till specifika enhetsteckensnitt beroende på uppspelningssystemet. Det är en bra rutin att ange standardnamn som dessa i alla teckensnittsbeskrivningar som använder enhetsteckensnitt. ”_serif” används som standard om `fontName` inte anges.

Egenskapen `fontPosture` kan ställas in på standardvärdet (`FontPosture.NORMAL`) eller kursiv stil (`FontPosture.ITALIC`). Egenskapen `fontWeight` kan ställas in på standardvärdet (`FontWeight.NORMAL`) eller på fet stil (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

Inbäddade teckensnitt jämfört med enhetsteckensnitt

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `fontLookup` i `FontDescription`-objektet anger om textmotorn söker efter ett enhetsteckensnitt eller ett inbäddat teckensnitt för textåtergivning. Om ett enhetsteckensnitt (`FontLookup.DEVICE`) anges, söker programmet efter teckensnittet på uppspelningssystemet vid körningstillfället. Om du anger ett inbäddat teckensnitt (`FontLookup.EMBEDDED_CFF`) söker programmet efter ett inbäddat teckensnitt i SWF-filen vid körningstillfället. Endast inbäddade CFF-teckensnitt (Compact Font Format) fungerar med den här inställningen. Ett reservteckensnitt på enheten används om det angivna teckensnitt inte hittas.

SWF-filens storlek blir mindre om enhetsteckensnitt används. Inbäddade teckensnitt ger enhetlig återgivning på olika plattformar.

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

Återgivningsläge och indikering

Flash Player 10 och senare, Adobe AIR 1.5 och senare

CFF-återgivning (Compact Font Format) är tillgängligt från och med Flash Player 10 och Adobe AIR 1.5. Den här typen av teckensnittsåtergivning gör texten tydligare och gör att små teckensnittsstorlekar kan visas med högre kvalitet. Inställningen gäller bara inbäddade teckensnitt. `FontDescription` ställs som standard in på den här inställningen (`RenderingMode.CFF`) för egenskapen `renderingMode` . Du kan ställa in egenskapen på `RenderingMode.NORMAL` så att den motsvarar återgivningstypen som används i Flash Player 7 eller tidigare versioner.

När CFF-återgivning är valt, styr en annan egenskap `cffHinting` hur vågräta linjer i glyferna passas in i delpixelrutnätet. Standardvärdet `CFFHinting.HORIZONTAL_STEM` använder CFF-indikering. Indikering tas bort om egenskapen ställs in på `CFFHinting.NONE`, vilket är lämpligt för animeringar och stora teckensnitt.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

Låsa och klona FontDescription

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När ett `FontDescription`-objekt tilldelas till ett `ElementFormat`, ställs egenskapen `locked` automatiskt in på `true`. Om du försöker ändra ett låst `FontDescription`-objekt, inträffar felet `IllegalOperationError`. Det bästa sättet är att helt definiera ett sådant objekt innan det tilldelas till en `ElementFormat`-instans.

Om du vill ändra en befintlig `FontDescription`-instans kontrollerar du först egenskapen `locked`. Om egenskapen är `true` kan du skapa en olåst kopia av objektet med metoden `clone()`. Du kan ändra egenskaperna för det olåsta objektet och sedan tilldela det till `ElementFormat`-instansen. Eventuella nya rader som skapas från detta `TextElement` kommer att ha den nya formateringen. Rader som skapats tidigare från samma objekt ändras inte.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Styra text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med FTE får du en ny uppsättning kontroller för textformatering som hanterar justering och mellanrum mellan tecken (kerning och spårning). Det finns även egenskaper som styr hur rader bryts och hur tabbstopp ställs in på rader.

Justera text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du justerar text får alla rader i ett stycke samma längd genom att avståndet mellan ord och ibland även mellan bokstäver justeras. Resultatet är att texten justeras på båda sidorna medan mellanrummen mellan ord och bokstäver varierar. Textkolumner i tidningar och tidskrifter är ofta justerade.

Med egenskapen `lineJustfication` i klassen SpaceJustifier kan du reglera justeringen av rader i ett textblock. Klassen LineJustification definierar konstanter som du kan använda för att ange ett justeringsalternativ: `ALL_BUT_LAST` justerar alla textrader utom den sista, `ALL_INCLUDING_LAST` justerar alla textrader inklusive den sista och `UNJUSTIFIED` (standardvärdet) justerar inte texten.

Om du vill justera text ska du ställa in egenskapen `lineJustification` på en instans av klassen SpaceJustifier och tilldela instansen till egenskapen `textJustifier` i en TextBlock-instans. I exemplet nedan skapas ett stycke i vilket alla textrader utom den sista är justerade.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new
SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

Ställ in egenskapen `SpaceJustifier.letterspacing` på `true` om du vill variera mellanrummet mellan bokstäverna samt mellanrummet mellan orden. Om mellanrum mellan bokstäver aktiveras kan fula mellanrum mellan ord elimineras som ibland kan inträffa med enkel justering.

Justera östasiatisk text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Det finns flera saker som du måste överväga när östasiatisk text justeras. Texten kan skrivas uppifrån och ned och vissa tecken, som kallas kinsoku, kan inte uppträda i början eller slutet av en rad. Klassen JustificationStyle definierar följande konstanter som anger alternativ hur sådana tecken hanteras. Med `PRIORITIZE_LEAST_ADJUSTMENT` justeras texten genom att raden expanderas eller komprimeras beroende på vilket som ger bäst resultat. Med `PUSH_IN_KINSOKU` justeras texten genom att raden komprimeras om det finns kinsoku i slutet av den och raden expanderas om kinsoku inte finns eller om utrymmet är otillräckligt.

Med `PUSH_OUT_ONLY` justeras texten genom att raden expanderas. Om du vill skapa ett block vertikal asiatisk text ska du ställa in egenskapen `TextBlock.lineRotation` på `TextRotation.ROTATE_90` och egenskapen `ElementFormat.textRotation` på `TextRotation.AUTO` som är standardvärdet. När egenskapen `textRotation` ställs in på `AUTO`, förblir glyferna i texten vertikala i stället för att vändas på sidan när raden roteras. Inställningen `AUTO` roterar endast breda glyfer och glyfer med full bredd 90 grader moturs så som anges av glyfens Unicode-egenskaper. I exemplet nedan visas ett vertikalt block med japansk text och texten justeras med alternativet `PUSH_IN_KINSOKU`.

```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
```

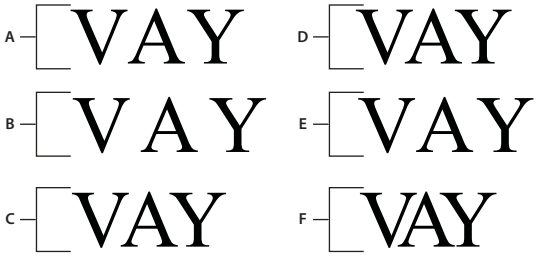
```
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

Kerning och spärra/knip

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Kerning och spårning påverkar mellanrummet mellan närliggande teckenpar i ett textblock. Kerning styr hur teckenpar ”passas ihop”, t.ex. paren ”WA” och ”Va”. Kerning ställs in i objektet `ElementFormat`. Det är aktiverat som standard (`Kerning.ON`) och kan ställas in på OFF eller AUTO, i vilket fall kerning bara tillämpas mellan tecken som inte är Kanji, Hiragana eller Katakana.

Spårning lägger till eller tar bort ett visst antal pixlar mellan alla tecken i ett textblock och det ställs också in i objektet `ElementFormat`. Det fungerar med både inbäddade teckensnitt och enhetsteckensnitt. FTE har stöd för två spårningsegenskaper, `trackingLeft` som lägger till/tar bort pixlar från tecknets vänstra sida, och `trackingRight` som lägger till/tar bort från tecknets högra sida. Om kerning ska användas, läggs spårningsvärdet till eller dras av kerningvärdena för varje teckenpar.



A. *Kerning.OFF* ***B.*** *TrackingRight=5, Kerning.OFF* ***C.*** *TrackingRight=-5, Kerning.OFF* ***D.*** *Kerning.ON* ***E.*** *TrackingRight=-5, Kerning.ON* ***F.*** *TrackingRight=-5, Kerning.ON*

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

Radbrytning för figursatt text

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Egenskapen `breakOpportunity` i objektet `ElementFormat` avgör vilka tecken som kan användas för brytning när figursatt text bryts upp i flera rader. Standardvärdet, `BreakOpportunity.AUTO`, använder Unicode standardegenskaper, t.ex. brytning mellan ord och bindestreck. Med `BreakOpportunity.ALL` kan vilket tecken som helst behandlas som en plats för eventuell radbrytning, vilket är praktiskt om du t.ex. vill skapa en effekt av text som slingrar sig längs en stig.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Tabbstopp

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du vill ange tabbstopp i ett textblock ska du definiera tabbstoppen genom att skapa instanser av klassen TabStop. Parametrarna för konstruktorn `TabStop()` anger hur text justeras med tabbstoppet. Parametrarna anger positionen för tabbstoppet och för decimaljusteringen, värdet som används för justering, uttryckt som en sträng. I vanliga fall är värdet en decimalpunkt, men det kan även vara ett komma, ett dollartecken, eller symbolen för yen eller euro till exempel. Koden nedan skapar ett tabbstopp kallat tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

När du har skapat tabbstoppen för textblocket ska du tilldela dem till egenskapen `tabStops` i en TextBlock-instans. Eftersom egenskapen `tabStops` kräver en vektor, ska du först skapa en vektor och sedan lägga till tabbstoppen. Med vektorn kan du tilldela en uppsättning tabbstopp till textblocket. Exemplet nedan skapar en `Vector<TabStop>`-instans och lägger till en uppsättning TabStop-objekt i den. Sedan tilldelas tabbstoppen till egenskapen `tabStops` i en TextBlock-instans.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Mer information om vektorer finns i ”Arbeta med arrayer” på sidan 24.

Exemplet nedan visar effekten av de olika justeringsalternativen för tabbstopp.

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDescription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Exempel på Flash Text Engine: nyhetslayout

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Programmeringsexemplet visar hur Flash-textmotorn används när en enkel tidningslayout skapas. Sidan innehåller en stor rubrik, en underrubrik och brödtext med flera kolumner.

Skapa först en FLA-fil och lägg till följande kod till ram 2 i standardlagret:

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as är kontrollskript i det här exemplet. Det anger innehåll, läser in formatinformation från en extern formatmall och tilldelar formaten till ElementFormat-objekt. Sedan skapas elementen för rubriken, underrubriken och brödtexten med flera kolumner.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;
```

```
        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the
Flash Text Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should
consider him a friend to mankind; and as our acquaintance commenced on that ground, it would
have been more agreeable to me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament
against the French Revolution and the National Assembly, I was in Paris, and had written to him
but a short time before to inform him how prosperously matters were going on. Soon after this I
saw his advertisement of the Pamphlet he intended to publish: As the attack was to be made in a
language but little studied, and less understood in France, and as everything suffers by
translation, I promised some of the friends of the Revolution in that country that whenever Mr.
Burke's Pamphlet came forth, I would answer it. This appeared to me the more necessary to be
done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and that
while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is
an imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the
circumstances I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence
in the world, and that some other mode might be found out to settle the differences that should
occasionally arise in the neighbourhood of nations. This certainly might be done if Courts were
disposed to set honesty about it, or if countries were enlightened enough not to be made the
dupes of Courts. The people of America had been bred up in the same prejudices against France,
which at that time characterised the people of England; but experience and an acquaintance with
the French Nation have most effectually shown to the Americans the falsehood of those prejudices;
and I do not believe that a more cordial and confidential intercourse exists between any two
countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int =
10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
```

```
        h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
        h2_ElFormat = getElFormat("h2", this.sheet);
        p_ElFormat = getElFormat("p", this.sheet);
        displayText();
    }

    public function drawBackground():void
    {
        var h:Number = this.storyTxt.y + this.storyTxt.height +
                        this.paddingTop + this.paddingBottom;
        var g:Graphics = this.graphics;
        g.beginFill(this.bgColor);
        g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
        g.endFill();
    }

    /**
     * Reads a set of style properties for a named style and then creates
     * a TextFormat object that uses the same properties.
     */
    public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
    {
    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);
    }
            var fd:FontDescription = new FontDescription(
                                style.fontFamily,
                                style.fontWeight,
                                FontPosture.NORMAL,
                                FontLookup.DEVICE,
                                RenderingMode.NORMAL,
                                CFFHinting.NONE);
    var format:ElementFormat = new ElementFormat(fd,
                          style.fontSize,
                          style.color,
                          1,
                          TextRotation.AUTO,
                          TextBaseline.ROMAN,
                          TextBaseline.USE_DOMINANT_BASELINE,
                          0.0,
                          Kerning.ON,
                          0.0,
                          0.0,
                          "en",
                                  BreakOpportunity.AUTO,
                                  DigitCase.DEFAULT,
                                  DigitWidth.DEFAULT,
                          LigatureLevel.NONE,
                                  TypographicCase.DEFAULT);

    if (style.hasOwnProperty("letterSpacing"))
    {
```

```
            format.trackingRight = style.letterSpacing;
        }
        }
        return format;
        }

        public function displayText():void
        {
        headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
        headlineTxt.x = this.paddingLeft;
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as används som en basklass när textblock skapas. Den inkluderar även verktygsfunktioner för att ändra teckensnittsstorlek och skiftläge.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
```

```
        tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
        te = new TextElement(blockText,this.ef1);
        tb.content = te;
        this.breakLines();
    }

    private function breakLines()
    {
        var textLine:TextLine = null;
        var y:Number = 0;
        var lineNum:int = 0;
        while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
        {
            textLine.x = 0;
            textLine.y = y;
            y += this.leading*textLine.height;
            this.addChild(textLine);
        }
        for (var i:int = 0; i < this.numChildren; i++)
        {
            TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
        }
        this.totalTextLines = this.numChildren;
    }

    private function rebreakLines()
    {
        this.clearLines();
        this.breakLines();
    }

    private function clearLines()
    {
        while(this.numChildren)
        {
            this.removeChildAt(0);
        }
    }
```

```
        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as utökar klassen FormattedTextBlock och används för att skapa rubriker. Den inkluderar en funktion för att passa in text i ett definierat utrymme på sidan.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                            Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > 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.totalTextLines > maxLines)
            {
                // set it back to the last size
                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.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as hanterar textformatering i en layout med flera kolumner. Det visar den flexibla användningen av ett TextBlock-objekt för att skapa, formatera och placera textrader.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
```

```
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

Kapitel 23: Använda Text Layout Framework

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Översikt över TLF (Text Layout Framework)

Flash Player 10 och senare, Adobe AIR 1.5 och senare

TLF (Text Layout Framework) är ett utökningsbart ActionScript-bibliotek. TLF bygger på den textmotor som finns i Adobe® Flash® Player 10 och Adobe® AIR® 1.5. TLF tillhandahåller avancerade funktioner för typografi och textlayout för webben. TLF kan användas med Adobe® Flex® och Adobe® Flash® Professional. Utvecklare kan använda eller utöka befintliga komponenter eller använda ramverket för att skapa egna textkomponenter.

TLF innehåller följande funktioner:

- Dubbelriktad text, lodrät text och fler än 30 skriftspråk, bland annat arabiska, hebreiska, kinesiska, japanska, koreanska, thailändska, laotiska och vietnamesiska
- Markering, redigering och flöde av text över flera kolumner och länkare behållare
- Lodrät text, Tate-Chu-Yoko (vågrät text i lodrät text) och justering för östasiatisk typografi
- Omfattande typografiska kontroller, bland annat kerning, ligaturer, skiftlägestypografi, skiftläge för siffror, sifferbredd och mjuka bindestreck
- Standardgester för tangentbord och möss för redigering, som klipp ut, kopiera, klistra in och ångra
- Avancerade utvecklar-API:er för att hantera textinnehåll, layout och kod samt skapa anpassade textkomponenter
- Robust liststöd med anpassade markörer och numreringsformat
- Infogade bilder och placeringsregler

Text Layout Framework är ett ActionScript 3.0-bibliotek som bygger på den textmotor (FTE) som introducerades i Flash Player 10. FTE kan öppnas via paketet `flash.text.engine` som ingår i Flash Player 10 Application Programming Interface (API).

Flash Player API:n ger åtkomst till textmotorn på låg nivå vilket innebär att vissa åtgärder kan kräva ganska stora mängder kod. TLF sammanför lågnivåkod i enklare API:er. TLF har också en begreppsmässig arkitektur, som ordnar de grundläggande beståndsdelarna som definieras i FTE till ett system som är enklare att använda.

TLF är till skillnad från FTE inte inbyggt i Flash Player. Det är snarare ett oberoende komponentbibliotek som är skrivet helt i ActionScript 3.0. Eftersom ramverket är utökningsbart kan det anpassas för olika miljöer. Både Flash Professional och Flex SDK innehåller komponenter som baseras på TLF-ramverket.

Fler hjälpavsnitt

"Flow" (ett TLF Markup-program)

Stöd för komplicerade skriftspråk

Flash Player 10 och senare, Adobe AIR 1.5 och senare

TLF har stöd för komplicerade skriftspråk. Stödet för komplicerade skriftspråk inkluderar möjligheten att visa och redigera text som skrivs från höger till vänster. Med TLF går det även att visa och redigera en blandning av skriftspråk som skrivs från vänster till höger eller från höger till vänster, som arabiska och hebreiska. Ramverket stöder inte bara lodrät textlayout för kinesiska, japanska och koreanska, utan även tate-chu-yoko (TCY-element). TCY-element är block av vågrät text inbäddat i körningar av lodrät text. Följande skriftspråk stöds:

- Latinska (engelska, spanska, franska, vietnamesiska och så vidare)
- Grekiska, kyrilliska, armeniska, georgiska och etiopiska
- Arabiska och hebreiska
- Han-ideografer och kana (kinesiska, japanska och koreanska) och hangul johab (koreanska)
- Thailändska, laotiska och khmer
- Devanagari, bengali, gurmukhi, malayalam, telugu, tamil, gujarati, oriya, kannada och tibetanska
- Tifinagh, yi, cherokee, canadian syllabics, deseret, shavian, vai, tagalog, hanunoo, buhid och tagbanwa

Använda Text Layout Framework i Flash Professional och Flex

Du kan använda TLF-klasserna direkt för att skapa anpassade komponenter i Flash. Dessutom innehåller Flash Professional CS5 en ny klass, fl.text.TLFTextField, som sammanför TLF-funktionerna. Använd klassen TLFTextField för att skapa textfält, som använder de avancerade textvisningsfunktionerna i TLF, i ActionScript. Skapa ett TLFTextField-objekt på samma sätt som du skapar ett textfält med klassen TextField. Använd sedan egenskapen `textFlow` för att tilldela avancerad formatering från TLF-klasserna.

Du kan också använda Flash Professional för att skapa TLFTextField-instansen på scenen med textverktyget. Sedan kan du använda ActionScript för att styra formateringen och layouten för textfältsinnehållet med TLF-klasserna. Du hittar mer information i avsnittet om TLFTextFieldi Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Om du arbetar i Flex använder du TLF-klasserna. Mer information finns i ”Använda Text Layout Framework” på sidan 427.

Använda Text Layout Framework

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Om du arbetar i Flex eller bygger anpassade textkomponenter använder du TLF-klasserna. TLF är ett ActionScript 3.0-bibliotek som ingår i sin helhet i biblioteket textLayout.swc. TLF-biblioteket innehåller cirka hundra ActionScript 3.0-klasser och gränssnitt som ordnats i tio paket. De här paketen ingår som underpaket i flashx.textLayout-paketet.

Klasserna i TLF (Text Layout Framework)

Flash Player 10 och senare, Adobe AIR 1.5 och senare

TLF-klasserna kan delas in i tre kategorier:

- Datastrukturer och formateringsklasser

- Återgivningsklasser
- Klasser för användarinteraktion

Datastrukturer och formateringsklasser

Följande paket innehåller datastrukturer och formateringsklasser för TLF:

- flashx.textLayout.elements
- flashx.textLayout.formats
- flashx.textLayout.conversion

Den huvudsakliga datastrukturen i TLF är textflödeshierarkin, som definieras i elements-paketet. I den här strukturen kan du tilldela format och attribut till textmängder med formats-paketet. Du kan även styra hur text importeras till, och exporteras från, datastrukturen med konverteringspaketet.

Återgivningsklasser

Följande paket innehåller återgivningsklasserna för TLF:

- flashx.textLayout.factory
- flashx.textLayout.container
- flashx.textLayout.compose

Klasserna i de här paketen underlättar textåtergivning för visning med Flash Player. Fabrikspaketet tillhandahåller ett enkelt sätt att visa statisk text. Behållarpaketet inkluderar klasser och gränssnitt som definierar visningsbehållare för dynamisk text. Kompositionspaketet definierar tekniker för positionering och visning av dynamisk text i behållare.

Klasser för användarinteraktion

Följande paket innehåller klasser för användarinteraktion för TLF:

- flashx.textLayout.edit
- flashx.textLayout.operations
- flashx.textLayout.events

Redigerings- och åtgärdspaketen definierar klasser som du kan använda för att tillåta redigering av text som lagras i datastrukturen. Händelsepaketet innehåller händelsehanteringsklasser.

Allmänna steg för att skapa text med TLF (Text Layout Framework)

Följande steg beskriver den allmänna arbetsgången för att skapa text med TLF:

1. Importera formaterad text till TLF-datastrukturerna. Du hittar mer information i ”Strukturera text med TLF” på sidan 432 och ”Formatera text med TLF” på sidan 436.
2. Skapa en eller flera länkade behållare för visningsobjekt för texten. Du hittar mer information i ”Hantera textbehållare med TLF” på sidan 437.
3. Koppla texten i datastrukturen till behållarna och ange alternativ för redigering och rullning. Du hittar mer information i ”Aktivera markering, redigering och ångra för text med TLF” på sidan 439.
4. Skapa en händelsehanterare för att flöda om texten som svar på en storleksändringshändelse (eller annan händelse). Du hittar mer information i ”Händelsehantering med TLF” på sidan 439.

Exempel på Text Layout Framework: nyhetslayout

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Följande exempel visar hur du använder TLF för att skapa layouten för en enkel tidningssida. Sidan har en stor rubrik, en underrubrik och ett avsnitt med brödtext i flera kolumner:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.geom.Rectangle;

    import flashx.textLayout.compose.StandardFlowComposer;
    import flashx.textLayout.container.ContainerController;
    import flashx.textLayout.container.ScrollPolicy;
    import flashx.textLayout.conversion.TextConverter;
    import flashx.textLayout.elements.TextFlow;
    import flashx.textLayout.formats.TextLayoutFormat;

    public class TLFNewsLayout extends Sprite
    {
        private var hTextFlow:TextFlow;
        private var headContainer:Sprite;
        private var headlineController:ContainerController;
        private var hContainerFormat:TextLayoutFormat;

        private var bTextFlow:TextFlow;
        private var bodyTextContainer:Sprite;
        private var bodyController:ContainerController;
        private var bodyTextContainerFormat:TextLayoutFormat;

        private const headlineMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008'><flow:p textAlign='center'><flow:span
fontFamily='Helvetica' fontSize='18'>TLF News Layout Example</flow:span><flow:br/><flow:span
fontFamily='Helvetica' fontSize='14'>This example formats text like a newspaper page with a
headline, a subtitle, and multiple columns</flow:span></flow:p></flow:TextFlow>";

        private const bodyMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008' fontSize='12' textIndent='10' marginBottom='15'
paddingTop='4' paddingLeft='4'><flow:p marginBottom='inherit'><flow:span>There are many
</flow:span><flow:span fontStyle='italic'>such</flow:span><flow:span> lime-kilns in that tract
of country, for the purpose of burning the white marble which composes a large part of the
substance of the hills. Some of them, built years ago, and long deserted, with weeds growing in
the vacant round of the interior, which is open to the sky, and grass and wild-flowers rooting
themselves into the chinks of the stones, look already like relics of antiquity, and may yet be
overspread with the lichens of centuries to come. Others, where the lime-burner still feeds his
daily and nightlong fire, afford points of interest to the wanderer among the hills, who seats
himself on a log of wood or a fragment of marble, to hold a chat with the solitary man. It is a
lonesome, and, when the character is inclined to thought, may be an intensely thoughtful
occupation; as it proved in the case of Ethan Brand, who had mused to such strange purpose, in
days gone by, while the fire in this very kiln was burning.</flow:span></flow:p><flow:p
marginBottom='inherit'><flow:span>The man who now watched the fire was of a different order, and
troubled himself with no thoughts save the very few that were requisite to his business. At
frequent intervals, he flung back the clashing weight of the iron door, and, turning his face
```

```
from the insufferable glare, thrust in huge logs of oak, or stirred the immense brands with a
long pole. Within the furnace were seen the curling and riotous flames, and the burning marble,
almost molten with the intensity of heat; while without, the reflection of the fire quivered on
the dark intricacy of the surrounding forest, and showed in the foreground a bright and ruddy
little picture of the hut, the spring beside its door, the athletic and coal-begrimed figure of
the lime-burner, and the half-frightened child, shrinking into the protection of his father's
shadow. And when again the iron door was closed, then reappeared the tender light of the half-
full moon, which vainly strove to trace out the indistinct shapes of the neighboring mountains;
and, in the upper sky, there was a flitting congregation of clouds, still faintly tinged with the
rosy sunset, though thus far down into the valley the sunshine had vanished long and long
ago.</flow:span></flow:p></flow:TextFlow>";

        public function TLFNewsLayout()
        {
            //wait for stage to exist
            addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        }

        private function onAddedToStage(evtObj:Event):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            // Headline text flow and flow composer
            hTextFlow = TextConverter.importToFlow(headlineMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // initialize the headline container and controller objects
            headContainer = new Sprite();
            headlineController = new ContainerController(headContainer);
            headlineController.verticalScrollPolicy = ScrollPolicy.OFF;
            hContainerFormat = new TextLayoutFormat();
            hContainerFormat.paddingTop = 4;
            hContainerFormat.paddingRight = 4;
            hContainerFormat.paddingBottom = 4;
            hContainerFormat.paddingLeft = 4;

            headlineController.format = hContainerFormat;
            hTextFlow.flowComposer.addController(headlineController);
            addChild(headContainer);
            stage.addEventListener(flash.events.Event.RESIZE, resizeHandler);

            // Body text TextFlow and flow composer
            bTextFlow = TextConverter.importToFlow(bodyMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // The body text container is below, and has three columns
            bodyTextContainer = new Sprite();
            bodyController = new ContainerController(bodyTextContainer);
            bodyTextContainerFormat = new TextLayoutFormat();
            bodyTextContainerFormat.columnCount = 3;
            bodyTextContainerFormat.columnGap = 30;

            bodyController.format = bodyTextContainerFormat;
            bTextFlow.flowComposer.addController(bodyController);
            addChild(bodyTextContainer);
```

```
            resizeHandler(null);
        }

        private function resizeHandler(event:Event):void
        {
            const verticalGap:Number = 25;
            const stagePadding:Number = 16;
            var stageWidth:Number = stage.stageWidth - stagePadding;
            var stageHeight:Number = stage.stageHeight - stagePadding;
            var headlineWidth:Number = stageWidth;
            var headlineContainerHeight:Number = stageHeight;

            // Initial compose to get height of headline after resize
            headlineController.setCompositionSize(headlineWidth,
headlineContainerHeight);
            hTextFlow.flowComposer.compose();
            var rect:Rectangle = headlineController.getContentBounds();
            headlineContainerHeight = rect.height;

            // Resize and place headline text container
            // Call setCompositionSize() again with updated headline height
            headlineController.setCompositionSize(headlineWidth, headlineContainerHeight );
            headlineController.container.x = stagePadding / 2;
            headlineController.container.y = stagePadding / 2;
            hTextFlow.flowComposer.updateAllControllers();

            // Resize and place body text container
            var bodyContainerHeight:Number = (stageHeight - verticalGap -
headlineContainerHeight);
            bodyController.format = bodyTextContainerFormat;
            bodyController.setCompositionSize(stageWidth, bodyContainerHeight );
            bodyController.container.x = (stagePadding/2);
            bodyController.container.y = (stagePadding/2) + headlineContainerHeight +
verticalGap;
            bTextFlow.flowComposer.updateAllControllers();
        }
    }
}
```

Klassen TLFNewsLayout använder två textbehållare. Den ena behållaren visar rubriken och underrubriken, och den andra visar brödtexten i tre kolumner. För enkelhetens skull är texten hårdkodad som TLF-kod i exemplet. Variabeln `headlineMarkup` innehåller både rubriken och underrubriken, och variabeln `bodyMarkup` innehåller brödtexten. Mer information om TLF-kod finns i ”Strukturera text med TLF” på sidan 432.

Efter en viss initiering importerar funktionen `onAddedToStage()` rubriktexten till ett TextFlow-objekt, som är huvuddatastrukturen i TLF:

```
hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);
```

Därefter skapas ett Sprite-objekt för behållaren, och en kontrollenhet skapas och kopplas till behållaren:

```
headContainer = new Sprite();
headlineController = new ContainerController(headContainer);
```

Kontrollenheten initieras för att ställa in alternativ för bland annat formatering och rullning. Kontrollenheten innehåller geometri, som definierar gränserna för den behållare till vilken texten flödar. Ett TextLayoutFormat-objekt innehåller formateringsalternativen:

```
hContainerFormat = new TextLayoutFormat();
```

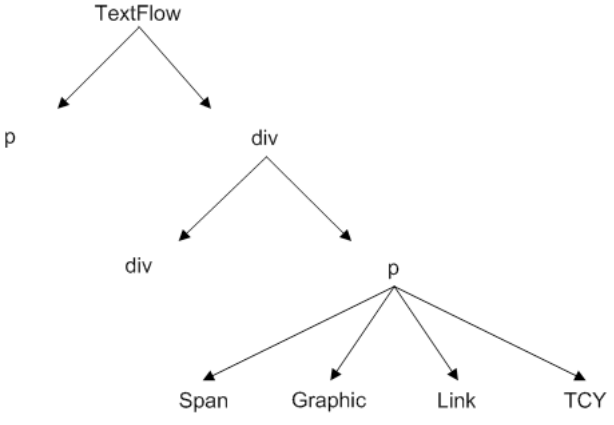
Kontrollenheten tilldelas till flödesdispositionen och funktionen lägger till behållaren i visningslistan. Dispositionen och visningen av behållarna anges däremot av metoden `resizeHandler()`. Samma stegsekvens utförs för att initiera TextFlow-objektet för brödtexten.

Metoden `resizeHandler()` mäter det tillgängliga utrymmet för återgivning av behållare och behållarnas storlek bestäms därefter. Efter ett inledande anrop till metoden `compose()` beräknas den faktiska höjden på rubrikens behållare. Metoden `resizeHandler()` kan sedan placera och visa rubrikbehållaren med metoden `updateAllControllers()`. Slutligen använder metoden `resizeHandler()` storleken på rubrikbehållaren för att avgöra placeringen av brödtextbehållaren.

Strukturera text med TLF

I TLF används ett hierarkiskt träd för att representera text. Varje nod i trädet är en instans av en klass som definieras i elementpaketet. Trädets rotnod är t.ex. alltid en instans av klassen TextFlow. Klassen TextFlow representerar en hel textartikel. En artikel är en samling text och andra element som behandlas som en enhet, eller ett flöde. En artikel kanske kräver mer än en kolumn eller en textbehållare för att kunna visas.

Med undantag för rotnoden är de övriga elementen löst baserade på XHTML-element. Följande diagram visar ramverkets hierarki:



Hierarkin TextFlow

TLF-kod

Det kan också vara bra att ha en förståelse för TLF-strukturen när du arbetar med TLF-kod. TLF-koden är en XML-representation av text som är en del av Text Layout Framework. Även om ramverket har stöd för andra XML-format är TLF-koden unik, eftersom den är baserad på strukturen i just TextFlow-hierarkin. Om du exporterar XML från en TextFlow-hierarki i det här formatet exporteras XML-koden med hierarkin oförändrad.

TLF-kod ger trognaste återgivningen av text i en TextFlow-hierarki. TLF-koden innehåller taggar för vart och ett av TextFlow-hierarkins grundläggande element, och den innehåller även attribut för alla formateringsegenskaper som finns i klassen TextLayoutFormat.

Följande tabell innehåller de taggar som kan användas i TLF Markup.

Element	Beskrivning	Underordnad	Klass
textflow	Rotelementet i koden.	div, p	TextFlow
div	En division inom ett TextFlow. Kan innehålla en grupp av flera stycken.	div, list, p	DivElement
p	Ett stycke.	a, tcy, span, img, tab, br, g	ParagraphElement
a	En länk.	tcy, span, img, tab, br, g	LinkElement
tcy	Ett flöde med vågrät text (används i ett lodrätt TextFlow).	a, span, img, tab, br, g	TCYElement
span	En körning av text inom ett stycke.		SpanElement
img	En bild i ett stycke.		InlineGraphicElement
tab	Ett tabbtecken.		TabElement
br	En radbrytning. Används för en radbrytning inom ett stycke. Texten fortsätter på nästa rad i samma stycke.		BreakElement
linkNormalFormat	Anger de formateringsattribut som används för länkar i normalt tillstånd.	TextLayoutFormat	TextLayoutFormat
linkActiveFormat	Anger de formateringsattribut som används för länkar i aktivt tillstånd, när musen klickar på en länk.	TextLayoutFormat	TextLayoutFormat
linkHoverFormat	Anger de formateringsattribut som används för länkar i hovringstillstånd, när musen befinner sig inom en länks gränser (rullar över den).	TextLayoutFormat	TextLayoutFormat
li	Ett listobjektselement. Måste finnas i ett listelement.	div, li, list, p	ListItemElement
list	En lista. Listor kan vara kapslade eller placeras intill varandra. Olika system för etiketter och numrering kan tillämpas på listobjekten.	div, li, list, p	ListElement
g	Ett gruppelement. Används för att gruppera element i ett stycke. Du kan kapsla in element under styckenivån.	a, tcy, span, img, tab, br, g	SubParagraphGroupElement

Fler hjälpavsnitt

TLF 2.0 listkod

TLF 2.0 SubParagraphGroupElements och typeName

Använda numrerade listor och punktlistor

Du kan använda klasserna ListElement och ListItemElement om du vill lägga till punktlistor i textkontrollerna. Punktlistorna kan vara kapslade och kan anpassas att använda olika punktformat (eller markörer) samt automatisk och konturerad numrering.

Använd taggen `<list>` om du vill skapa listor i textflödena. Sedan använder du taggarna `<li>` i taggen `<list>` för varje listobjekt i listan. Du kan anpassa punkternas utseende med hjälp av klassen ListMarkerFormat.

Med följande exempel skapas enkla listor:

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Du kan kapsla in listor i andra listor enligt följande exempel:

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:list paddingRight="24" paddingLeft="24">
        <flow:li>Item 1a</flow:li>
        <flow:li>Item 1b</flow:li>
        <flow:li>Item 1c</flow:li>
    </flow:list>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Om du vill anpassa typen av markör i listan använder du egenskapen `listStyleType` för ListElement. Den här egenskapen kan vara alla värden som definieras av klassen ListStyleType (till exempel `check`, `circle`, `decimal` och `box`). I följande exempel skapas listor med olika markörtyper och en anpassad numreringsökning:

```
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperAlpha">
<flow:li>upperAlpha item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list
paddingRight="24" paddingLeft="24" listStyleType="lowerAlpha"> <flow:li>lowerAlpha
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24"
paddingLeft="24" listStyleType="upperRoman"> <flow:li>upperRoman item</flow:li>
<flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerRoman"> <flow:listMarkerFormat> <!-- Increments the list by 2s rather than
1s. --> <flow:ListMarkerFormat counterIncrement="ordered 2"/> </flow:listMarkerFormat>
<flow:li>lowerRoman item</flow:li> <flow:li>another</flow:li> </flow:list>
```

Du kan använda klassen ListMarkerFormat om du vill definiera numreringen. Förutom att definiera numreringens ökning kan du även anpassa numreringen genom att återställa den med egenskapen `counterReset`.

Du kan ytterligare anpassa markörernas utseende i listorna med hjälp av egenskaperna `beforeContent` och `afterContent` för ListMarkerFormat. De här egenskaperna gäller innehåll som visas före och efter markörens innehåll.

I följande exempel läggs strängen "XX" till före markören och strängen "YY" efter markören:

```
<flow:list listStyleType="upperRoman" paddingLeft="36" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="XX"
            afterContent="YY"
            counterIncrement="ordered -1"/>
        </flow:listMarkerFormat>
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Egenskapen `content` kan också användas för att anpassa markörformatet. I följande exempel visas en ordnad markör med romersk-numeriskt format i versaler:

```
<flow:list listStyleType="disc"  paddingLeft="96" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="Section "
            content="counters(ordered,&quot;*&quot;,upperRoman)"
            afterContent=": "/>
    </flow:listMarkerFormat>
    <flow:li>Item 1</li>
    <flow:li>Item 2</li>
    <flow:li>Item 3</li>
</flow:list>
```

Som exemplet ovan visar kan egenskapen `content` även infoga ett suffix: En sträng som visas efter markören men före `afterContent`. Om du vill infoga den här strängen när du lägger till XML-innehåll i flödet ska du omsluta strängen med `&quote;` HTML-entiteter i stället för citattecken(`"<sträng>"`).

Fler hjälpavsnitt

TLF 2.0 listkod

Använda utfyllnad i TLF

Varje FlowElement har stöd för utfyllnadsegenskaper som du använder om du vill kontrollera placeringen av varje elements innehållsområde samt avståndet mellan innehållsområdena.

Ett elements totala bredd är summan av innehållets bredd plus egenskaperna `paddingLeft` och `paddingRight`. Ett elements totala höjd är summan av innehållets höjd plus egenskaperna `paddingTop` och `paddingBottom`.

Utfyllnaden är avståndet mellan kanten och innehållet. Utfyllnadsegenskaperna är `paddingBottom`, `paddingTop`, `paddingLeft` och `paddingRight`. Utfyllnad kan användas på TextFlow-objektet samt på följande underordnade element:

- div
- img
- li
- list
- p

Utfyllnadsegenskaper kan inte användas på span-element.

Med följande exempel anges utfyllnadsegenskaper på TextFlow:

```
<flow:TextFlow version="2.0.0" xmlns:flow="http://ns.adobe.com/textLayout/2008" fontSize="14"
textIndent="15" paddingTop="4" paddingLeft="4" fontFamily="Times New Roman">
```

Giltiga värden för utfyllnadsegenskaper är tal (i pixlar), ”auto” eller ”inherit”. Standardvärdet är "auto", vilket betyder att det beräknas automatiskt och anges till 0 för alla element utom ListElement. För ListElement är "auto" 0 utom på startsidan av listan där värdet för egenskapen `listAutoPadding` används. Standardvärdet för `listAutoPadding` är 40, vilket ger listor ett standardindrag.

Utfyllnadsegenskaperna ärvs inte som standard. Värdena "auto" och "inherit" är konstanter som definieras av klassen FormatValue.

Utfyllnadsegenskaper kan vara negativa värden.

Fler hjälpavsnitt

Utfyllnadsändringar i TLF 2.0

Formatera text med TLF

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Paketet flashx.textLayout.formats innehåller gränssnitt och klasser, som du kan använda för att tilldela format till alla FlowElement i textflödeshierarkin. Du kan tillämpa formateringen på två sätt. Du kan tilldela ett specifikt format enskilt eller tilldela en grupp av format samtidigt med ett särskilt formateringsobjekt.

ITextLayoutFormat-gränsnittet innehåller alla format som kan tillämpas på ett FlowElement. Vissa format tillämpas på en hel textbehållare eller ett helt textstycke men tillämpas inte logiskt på enskilda tecken. Formatering som justering och tabbstopp tillämpas till exempel på hela stycken, men inte på enskilda tecken.

Tilldela format till ett FlowElement med egenskaper

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan ange format på ett FlowElement genom att tilldela egenskaper. FlowElement-klassen implementerar ITextLayoutFormat-gränssnittet, vilket innebär att alla underklasser av FlowElement-klassen också måste implementera det gränssnittet.

Följande kod är ett exempel på hur du tilldelar enskilda format till en ParagraphElement-instans:

```
var p:ParagraphElement = new ParagraphElement();
p.fontSize = 18;
p.fontFamily = "Arial";
```

Tilldela format till ett FlowElement med klassen TextLayoutFormat

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Du kan ange format på ett FlowElement med klassen TextLayoutFormat. Du kan använda den här klassen för att skapa ett särskilt formateringsobjekt som innehåller alla formateringsvärden som du vill använda. Du kan sedan tilldela objektet till egenskapen `format` i ett FlowElement-objekt. Både TextLayoutFormat och FlowElement implementerar ITextLayoutFormat-gränssnittet. Detta säkerställer att båda klasserna innehåller samma formategenskaper.

Du hittar mer information i avsnittet om TextLayoutFormat i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Ärva format

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Format ärvs genom textflödeshierarkin. Om du tilldelar en TextLayoutFormat-instans till en FlowElement-instans med underordnade, initierar ramverket en process som kallas *cascade*. Under en cascade undersöker ramverket rekursivt varje nod i hierarkin som ärvs från FlowElement. Sedan avgörs om de ärvda värdena ska tilldelas varje formateringsegenskap. Följande regler gäller för en cascade:

1 Egenskapsvärden ärvs bara från direkt överordnade.

2 Egenskapsvärden ärvs bara om egenskapen inte har ett värde (det vill säga om värdet är `undefined`)..

3 Vissa attribut ärver inte värden när de är odefinierade om inte attributets värde är inställt på ”inherit” eller konstanten `flashx.textLayout.formats.FormatValue.INHERIT`.

Om du t.ex. anger värdet `fontSize` på TextFlow-nivå, gäller inställningen alla element i textflödet. Värdet överförs med andra ord nedåt i textflödeshierarkin. Du kan dock åsidosätta värdet i ett visst element genom att tilldela elementet ett nytt värde direkt. Om du däremot anger värdet för `backgroundColor` på TextFlow-nivå ärvs inte TextFlow-värdet av de underordnade. Egenskapen `backgroundColor` ärvs inte från en överordnad under en cascade. Du kan åsidosätta detta beteende genom att ställa in egenskapen `backgroundColor` för varje underordnat objekt på: `flashx.textLayout.formats.FormatValue.INHERIT`.

Du hittar mer information i avsnittet om TextLayoutFormat i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Importera och exportera text med TLF

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Med klassen TextConverter i paketet flashx.textLayout.conversion.* kan du importera och exportera text till och från TLF. Använd den här klassen om du tänker läsa in text vid körningen i stället för att kompilera texten till SWF-filen. Du kan också använda den här klassen för att exportera text som lagras i en TextFlow-instans till ett String- eller XML-objekt.

Både import och export är enkla procedurer. Du anropar antingen metoden `export()` eller metoden `importToFlow()` som båda ingår i TextConverter-klassen. Båda metoderna är statiska vilket innebär att du anropar metoderna för TextConverter-klassen i stället för en instans av TextConverter-klassen.

Klasserna i paketet flashx.textLayout.conversion är mycket flexibla när det gäller var texten ska sparas. Om du till exempel lagrar texten i en databas kan du importera texten till ramverket för att visa den. Du kan sedan använda klasserna i paketet flashx.textLayout.edit för att ändra texten och exportera tillbaka den ändrade texten till databasen.

Du hittar mer information i avsnittet om flashx.textLayout.conversion i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Hantera textbehållare med TLF

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När texten väl har lagrats i TLF-datastrukturerna kan den visas med Flash Player. Texten som lagras i flödeshierarkin måste konverteras till ett format som Flash Player kan visa. Du kan skapa visningsobjekt från ett flöde på två sätt i TLF. Det första, enklare tillvägagångssättet passar bäst för att visa statisk text. På det andra, mer komplicerade sättet kan du skapa dynamiskt text, som kan markeras och redigeras. I båda fallen konverteras texten till slut till instanser av TextLine-klassen, som är en del av flash.text.engine.*-paketet i Flash Player 10.

Skapa statisk text

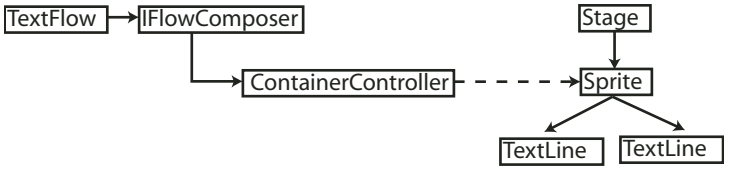
För det enkla sättet används klassen TextFlowTextLineFactory som finns i paketet flashx.textLayout.factory. Fördelen med metoden, förutom dess enkelhet, är att den kräver mindre minne än metoden FlowComposer. Den här metoden är lämplig för statisk text som användaren inte behöver redigera, markera eller bläddra.

Du hittar mer information i avsnittet om TextFlowTextLineFactory i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Skapa dynamisk text och behållare

Använd en flödeshanterare om du vill ha större kontroll över hur texten visas än vad som erbjuds i TextFlowTextLineFactory. Med en flödeshanterare kan användarna till exempel markera och redigera texten. Du hittar mer information i ”Aktivera markering, redigering och ångra för text med TLF” på sidan 439.

En flödeshanterare är en instans av klassen StandardFlowComposer i paketet flashx.textLayout.compose. En flödeshanterare hanterar konverteringen av TextFlow-instanser till TextLine-instanser samt placeringen av dessa TextLine-instanser i en eller flera behållare.



En IFlowComposer har noll eller fler ContainerControllers

Varje TextFlow-instans har ett motsvarande objekt som implementerar IFlowComposer-gränssnittet. IFlowComposer-objektet kan nås via egenskapen `TextFlow.flowComposer`. Du kan anropa metoder som definieras av IFlowComposer-gränssnittet via den här egenskapen. Med de här metoderna kan du koppa text till en eller flera behållare och förbereda texten för visning i en behållare.

En behållare är en instans av Sprite-klassen som är en underklass av klassen DisplayObjectContainer. Båda dessa klasser ingår i API:n för visningslista i Flash Player. En behållare är en mer avancerad form av markeringsramen som används med TextLineFactory-klassen. Precis som markeringsramen, anger en behållare det område där TextLine-instanser kan visas. Till skillnad från en markeringsram har en behållare ett motsvarande kontrollenhetsobjekt. Kontrollenheten hanterar rullning, disposition, länkning, formatering och händelsehantering för en behållare eller en grupp med behållare. Varje behållare har ett kontrollenhetsobjekt som är en instans av klassen ContainerController i paketet flashx.textLayout.container.

Om du vill visa text skapar du ett kontrollenhetsobjekt, som hanterar behållaren och koppar den till flödesdispositionen. När du har associerat behållaren komponerar du texten så att den kan visas. Behållare har alltså två lägen: disposition och visning. Disposition är processen att konvertera texten från textflödeshierarkin till TextLine-instanser och beräkna om instanserna passar in i behållaren. Visning är processen att uppdatera visningslistan i Flash Player.

Du hittar mer information i avsnitten om IFlowComposer, StandardFlowComposer och ContainerController i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Aktivera markering, redigering och ångra för text med TLF

Flash Player 9.0 och senare, Adobe AIR 1.0 och senare

Möjligheten att markera eller redigera text hanteras på textflödesnivå. Till varje instans av TextFlow-klassen finns en associerad interaktionshanterare. Du kommer åt interaktionshanteraren för TextFlow-objektet genom egenskapen `TextFlow.interactionManager` för objektet. Om du vill aktivera textmarkering tilldelar du en instans av klassen SelectionManager till egenskapen `interactionManager`. Om du vill aktivera både textmarkering och redigering tilldelar du en instans av klassen EditManager i stället för en instans av klassen SelectionManager. Om du vill göra det möjligt att ångra åtgärder skapar du en instans av klassen UndoManager och inkluderar den som ett argument i anropet till konstruktorn för EditManager. UndoManager-klassen lagrar en historik över användarens senaste redigeringsåtgärder så att användaren kan ångra eller göra om specifika redigeringsåtgärder. Alla dessa tre klasser hör till redigeringspaketet.

Du hittar mer information i avsnitten om SelectionManager, EditManager och UndoManager i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Händelsehantering med TLF

Flash Player 10 och senare, Adobe AIR 1.5 och senare

TextFlow-objekt skickar händelser i bland annat följande situationer:

- När texten eller layouten ändras
- Innan en åtgärd utförs och efter att den har slutförts
- När statusen för ett FlowElement-objekt ändras
- När en dispositionsåtgärd slutförs

Du hittar mer information i avsnittet om flashx.textLayout.events i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Fler hjälpavsnitt

TLF FlowElement- och LinkElement-händelser och EventMirrors

Placera bilder i text

Du kan placera InlineGraphicElement i texten med hjälp av följande egenskaper:

- `float`-egenskapen för klassen InlineGraphicElement
- `clearFloats`-egenskapen för FlowElement

Med egenskapen `float` styr du placeringen av grafiken och texten runtomkring. Med egenskapen `clearFloats` styr du placeringen av styckelementen i förhållande till `float`.

Om du vill kontrollera placeringen av en bild i ett textelement använder du egenskapen `float`. I följande exempel läggs en bild till i ett stycke och vänsterjusterar den så att texten omger bilden till höger:

```
<flow:p paragraphSpaceAfter="15" >Images in a flow are a good thing. For example, here is a
float. It should show on the left: <flow:img float="left" height="50" width="19"
source="../assets/bulldog.jpg"></flow:img> Don't you agree? Another sentence here. Another
sentence here. Another sentence here. Another sentence here. Another sentence here. Another
sentence here. Another sentence here. Another sentence here.</flow:p>
```

Giltiga värden för egenskapen `float` är "left", "right", "start", "end" och "none". Klassen Float definierar de här konstanterna. Standardvärdet är "none".

Egenskapen `clearFloats` är användbar om du vill justera startplaceringen för efterföljande stycken som normalt skulle omsluta bilden. Anta att du till exempel har en bild som är större än det första stycket. Du kan då se till att det andra stycket börjar *efter* bilden genom att ange egenskapen `clearFloats`.

I följande exempel används en bild som är högre än texten i det första stycket. För att se till att det andra stycket börjar efter bilden i textblocket anges egenskapen `clearFloats` i det andra stycket till "end".

```
<flow:p paragraphSpaceAfter="15" >Here is another float, it should show up on the right:
<flow:img float="right" height="50" elementHeight="200" width="19"
source="../assets/bulldog.jpg"></flow:img>We'll add another paragraph that should clear past
it.</flow:p><flow:p clearFloats="end" >This should appear after the previous float on the
right.</flow:p>
```

Giltiga värden för egenskapen `clearFloats` är "left", "right", "end", "start", "none" och "both". Klassen ClearFloats definierar konstanterna. Du kan även ange egenskapen `clearFloats` till "inherit", som är en konstant som definieras av klassen FormatValue. Standardvärdet är "none".

Fler hjälpavsnitt

TLF Floats

Kapitel 24: Arbeta med ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript är gjort för omslutande, interaktiva program; ljud är ett ofta förbisett element för kraftfulla omslutande program. Du kan lägga till ljudeffekter till ett videospel, rundgång i ett programgränssnitt för användare eller också göra ett program som analyserar MP3-filer som lästs in via Internet med ljud i kärnan av programmet.

Du kan läsa in externa ljudfiler och arbeta med ljud som är inbäddat i en SWF-fil. Du kan styra ljudet, skapa visuella representationer av ljudinformationen och spela in ljud från en mikrofon.

Fler hjälpavsnitt

flash.media-paket

flash.events.SampleDataEvent

Grunderna om att arbeta med ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Datorer kan fånga in och koda digitalt ljud – datorrepresentation av ljudinformation – och lagra och hämta det för att spela upp det via högtalare. Du kan spela upp ljud med Adobe® Flash® Player eller Adobe® AIR™ och ActionScript.

När ljuddata har konverterats till digitalt format, har det olika egenskaper som t.ex. ljudets volym och huruvida det är stereo- eller monoljud. När du spelar upp ett ljud i ActionScript, kan du justera dessa egenskaper också genom att t.ex. öka ljudvolymen eller få det att låta som om det kom från ett visst håll.

Innan du kan styra ett ljud i ActionScript måste du läsa in ljudinformationen i Flash Player eller AIR. Det finns fem sätt att läsa in ljuddata i Flash Player eller AIR så att du kan arbeta med det i ActionScript.

- Läs in en extern ljudfil, till exempel en mp3-fil, till SWF-filen.
- Bädda in ljudinformationen i SWF-filen direkt när den skapas.
- Hämta ljud från en mikrofon som är kopplad till en användares dator.
- Direktuppspela ljud från en server.
- Generera och spela upp ljud dynamiskt.

När du läser in ljuddata från en extern ljudfil kan du börja uppspelningen från början av ljudfilen, medan återstående ljuddata fortfarande läses in.

Även om det finns flera ljudfilsformat som används för att koda digitalt ljud, stöder ActionScript 3.0, Flash Player och AIR ljudfiler som lagras i MP3-formatet. Det går inte att direkt läsa in eller spela upp ljudfiler i andra format, som WAV eller AIFF.

När du arbetar med ljud i ActionScript, kommer du troligen att arbeta med flera klasser i flash.media-paketet. Klassen Sound är den klass som du använder för att komma åt ljudinformation genom att läsa in en ljudfil eller tilldela en funktion till en händelse som samplar ljuddata och sedan börjar uppspelningen. När du börjar spela upp ett ljud, får du åtkomst till ett SoundChannel-objekt i Flash Player och AIR. Eftersom ljudfilen som du läste in kanske bara är ett av flera ljud som spelas upp på användarens dator, använder varje enskilt ljud som spelas upp sitt eget SoundChannel-objekt. De kombinerade utdata från alla SoundChannel-objekt som blandas är det som faktiskt spelas upp i högtalarna på användarens dator. Du använder den här SoundChannel-instansen för att styra egenskaperna för ljudet och stoppa uppspelningen. Slutligen, om du vill styra det kombinerade ljudet sker det med klassen SoundMixer som ger dig kontroll över blandade utdata.

Du kan också använda flera andra klasser för att utföra mer specifika uppgifter när du arbetar med ljud i ActionScript. Mer information om alla ljudrelaterade klasser finns i ”Ljudarkitekturen” på sidan 442.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kanske kommer att stöta på:

Amplitud Avståndet för en punkt i ljudvågen från noll eller jämviktslinjen.

Bithastighet Den datamängd som kodas eller direktuppspelas för varje sekund i en ljudfil. För MP3-filer anges vanligtvis bithastigheten i tusen bitar per sekund (kbp/s). En högre bithastighet innebär vanligtvis en högre kvalitet på ljudvågen.

Buffring Mottagning och lagring av ljuddata innan de spelas upp.

mp3 MPEG-1 Audio Layer 3 eller MP3 är ett populärt ljudkomprimeringsformat.

Panorering Placeringen av en ljudsignal mellan vänster och höger kanal i ett stereoljudfält.

Topp Den högsta punkten i en vågform.

Samplingsfrekvens Definierar antalet mätvärden per sekund tagna från en analog ljudsignal för att skapa en digital signal. Samplingsfrekvensen för ljud på en standard-cd-skiva är 44,1 kHz eller 44 100 samplingar per sekund.

Direktuppspelning Processen att spela upp de första delarna av ljudfilen eller videofilen under tiden som senare delar av filen fortfarande läses in från en server.

Volym Styrkan hos ett ljud.

Vågform Formen på en graf över de varierande amplituderna för en ljudsignal över tiden.

Ljudarkitekturen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmen kan läsa in ljuddata från fem huvudkällor:

- Externa ljudfiler läses in vid körning
- Inbäddade ljudobjekt i programmets SWF-fil
- Ljuddata från en mikrofon som är ansluten till användarens system
- Ljuddata direktuppspelat från en fjärrmediaserver, t.ex. Flash Media-server
- Ljuddata skapas dynamiskt via händelsehanteraren `sampleData`.

Ljuddata kan antingen vara läsas in helt innan de spelas upp eller direktuppspelas, vilket betyder att de spelas upp medan inläsningen pågår.

Du kan använda ljudfiler som är lagrade i MP3-format i ActionScript 3.0-klasser. Det går inte att direkt läsa in eller spela upp ljudfiler i andra format som WAV eller AIFF. Från och med Flash Player 9.0.115.0 kan emellertid AAC-ljudfiler läsas in och spelas upp med klassen NetStream. Detta är samma teknik som används för att läsa in och spela upp videoinnehåll. Mer information om den här tekniken finns i ”Arbeta med video” på sidan 474.

I Adobe Flash Professional kan du importera WAV- eller AIFF-ljudfiler och bädda in dem i programmets SWF-filer i MP3-format. Med Flash-utvecklingsverktyget kan du komprimera inbäddade ljudfiler för att minska filstorleken men det försämrar ljudkvaliteten. Mer information finns i ”Importera ljud” i *Använda Flash*.

ActionScript 3.0 ljudarkitektur använder följande klasser i flash.media-paketet.

Klass	Beskrivning
flash.media.Sound	Klassen Sound hanterar inläsning av ljud och grundläggande ljudegenskaper samt startar en uppspelning av ljudet.
flash.media.SoundChannel	När programmet spelar upp ett Sound-objekt, skapas ett nytt SoundChannel-objekt för styrning av uppspelningen. SoundChannel-objektet styr ljudvolymen i både vänstra och högra uppspelningskanalerna. Varje ljud som spelas upp har sitt eget SoundChannel-objekt.
flash.media.SoundLoaderContext	I klassen SoundLoaderContext anges hur många sekunders buffring som ska användas vid inläsning av ljud och om Flash Player och AIR ska söka efter en principfil på servern när en fil läses in. Ett SoundLoaderContext-objekt används som parameter till metoden `Sound.load()`.
flash.media.SoundMixer	Klassen SoundMixer styr uppspelnings- och säkerhetsegenskaper som rör alla ljud i ett program. I själva verket så blandas flera ljudkanaler genom ett allmänt SoundMixer-objekt, vilket innebär att egenskapsvärden i SoundMixer-objektet påverkar alla SoundChannel-objekt som spelas upp just nu.
flash.media.SoundTransform	Klassen SoundTransform innehåller värden som styr ljudvolym och panorering. Ett SoundTransform-objekt kan användas på ett individuellt SoundChannel-objekt, på det globala SoundMixer-objektet eller ett Microphone-objekt bland annat.
flash.media.ID3Info	Ett ID3Info-objekt innehåller egenskaper som representerar ID3-metadatainformation som ofta lagras i MP3-ljudfiler.
flash.media.Microphone	Klassen Microphone representerar en mikrofon eller andra ljudindataenheter som anslutits till användarens dator. Ljudindata från en mikrofon kan dirigeras till lokala högtalare eller skickas till en fjärrserver. Objektet Microphone styr förstärkningen, samplingsfrekvensen och andra egenskaper i sitt eget ljudflöde.
flash.media.AudioPlaybackMode	Klassen AudioPlaybackMode definierar konstanter för audioPlaybackMode-egenskapen i SoundMixer-klassen.

Varje ljud som läses in och spelas upp behöver sin egen instans av klassen Sound och klassen SoundChannel. Utdata från flera SoundChannel-instanser blandas sedan av den globala klassen SoundMixer under uppspelningen.

Klasserna Sound, SoundChannel och SoundMixer används inte för ljuddata från en mikrofon eller vid direktuppspelning från en medieserver som Flash Media-server.

Läsa in externa ljudfiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Varje instans av klassen Sound finns för att läsa in och aktivera uppspelningen av ett specifikt ljudobjekt. Ett program kan inte återanvända ett Sound-objekt eller läsa in fler än ett ljud. Om ett nytt ljudobjekt ska läsas in måste ett nytt Sound-objekt skapas.

Om du läser in en liten ljudfil, t.ex. ett klickljud till en knapp, kan programmet skapa ett nytt Sound-objekt och automatiskt läsa in ljudfilen enligt exemplet nedan:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

`Sound()`-konstruktorn accepterar ett URLRequest-objekt som sin första parameter. När ett värde för URLRequest-parametern finns, börjar det nya Sound-objektet läsa in det specificerade ljudobjektet automatiskt.

I alla förutom de enklaste fallen, bör programmet bevaka ljudinläsningsprocessen för att upptäcka fel under inläsningen. Om ljudet är ganska stort kanske det inte har lästs in fullständigt när användaren klickar på knappen som aktiverar ljudet. Om du försöker spela upp ett ljud som inte är inläst kan det orsaka körningsfel. Det är säkrare att vänta tills ljudinläsningen är slutförd innan användarna får utföra åtgärder som kan starta uppspelning av ljudet.

Ett Sound-objekt skickar ett antal olika händelser under ljudinläsningsprocessen. Programmet kan avlyssna dessa händelser för att spåra inläsningsprocessen och säkerställa att ljudet är fullständigt inläst innan uppspelningen. Följande tabell visar vilka händelser som kan skickas av ett Sound-objekt.

Händelse	**Beskrivning**
open (`Event.OPEN`)	Skickas precis innan ljudinläsningsåtgärden startar.
progress (`ProgressEvent.PROGRESS`)	Skickas då och då under ljudinläsningsprocessen när data hämtas från filen eller strömmen.
id3 (`Event.ID3`)	Skickas när ID3-data finns tillgängliga för ett MP3-ljud.
complete (`Event.COMPLETE`)	Skickas när alla ljudresursdata är fullständigt inlästa.
ioError (`IOErrorEvent.IO_ERROR`)	Skickas när en ljudfil inte kan hittas eller när inläsningsprocessen avbryts innan alla ljuddata har mottagits.

Följande kod visar hur du spelar upp ett ljud när det är fullständigt inläst:

```
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();
}
```

Först skapas ett nytt Sound-objekt i kodexemplet utan att det får ett initialt värde för URLRequest-parametern. Sedan avlyssnar det `Event.COMPLETE`-händelsen från Sound-objektet som startar körningen av `onSoundLoaded()`-metoden när alla ljuddata är inlästa. Härefter anropas `Sound.load()`-metoden med ett nytt URLRequest-värde för ljudfilen.

Metoden `onSoundLoaded()` körs när ljudet är fullständigt inläst. Egenskapen `target` för Event-objektet är en referens till Sound-objektet. Sedan startas ljuduppspelningen med ett anrop till `play()`-metoden för Sound-objektet.

Övervaka ljudinläsningsprocessen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ljudfiler kan vara mycket stora och ta lång tid att läsa in. Eftersom du kan spela upp ljud i Flash Player och AIR innan det är fullständigt inläst, kanske du vill visa användaren hur mycket ljuddata som har lästs in och hur mycket av ljudet som redan har spelats upp.

Klassen Sound skickar två händelser som gör det relativt enkelt att visa inläsningsförloppet för ett ljud: `ProgressEvent.PROGRESS` och `Event.COMPLETE`. I följande exempel visas hur du använder dessa händelser för att visa information om förloppet för det ljud som läses in:

```
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);
}
```

Först skapar koden ett Sound-objekt och lägger sedan till avlyssnare i objektet för händelserna `ProgressEvent.PROGRESS` och `Event.COMPLETE`. När `Sound.load()`-metoden har anropats och första data har mottagits från ljudfilen, inträffar en `ProgressEvent.PROGRESS`-händelse som aktiverar `onSoundLoadProgress()`-metoden.

Procentsatsen för ljuddata som har lästs in är lika med värdet för `bytesLoaded`-egenskapen i ProgressEvent-objektet delat med värdet för `bytesTotal`-egenskapen. Samma `bytesLoaded`- och `bytesTotal`-egenskaper finns också tillgängliga i Sound-objektet. I exemplet ovan visas endast meddelanden om ljudinläsningsförloppet, men du kan enkelt använda `bytesLoaded`- och `bytesTotal`-värdena för uppdatering av komponenter i förloppsindikatorn, t.ex. de som finns i Adobe Flex Framework eller Adobe Flash-utvecklingsverktyget.

I det här exemplet visas även hur ett program kan identifiera och svara på ett fel under inläsning av ljudfiler. Om t.ex. en ljudfil med ett visst filnamn inte kan hittas, skickar Sound-objektet en `Event.IO_ERROR`-händelse. I föregående kod körs `onIOError()`-metoden och visar ett kort felmeddelande när ett fel uppstår.

Arbeta med inbäddade ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Inbäddade ljud i stället för ljud som lästs in från en extern fil är mycket användbara när det gäller små ljud som används som indikatorer i programmets användargränssnitt, t.ex. ljud som spelas upp när du klickar på en knapp.

När du bäddar in en ljudfil i programmet, ökar storleken på den slutliga SWF-filen med storleken på ljudfilen. M.a.o. kan SWF-filen växa till oönskad storlek när du bäddar in stora ljudfiler i programmet.

Den exakta metoden att bädda in en ljudfil i programmets SWF-fil varierar beroende på utvecklingsmiljön.

Använda en inbäddad fil i Flash

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med Flash-utvecklingsverktyget kan du importera ljud i ett antal ljudformat och lagra dem som symboler i biblioteket. Sedan kan du koppla dem till bildrutor på tidslinjen eller till bildrutorna för ett knappläge, använda dem med beteenden eller använda dem direkt i ActionScript-koden. I det här avsnittet beskrivs hur du använder inbäddade ljud i ActionScript-kod med Flash-utvecklingsverktyget. Mer information om andra sätt att använda inbäddade ljud i Flash finns i ”Importera ljud” i *Använda Flash*.

Så här bäddar du in en ljudfil med Flash-utvecklingsverktyget:

1. Välj Arkiv > Importera > Importera till bibliotek, välj en ljudfil och importera den.
2. Högerklicka på namnet för den importerade filen i bibliotekspanelen och välj Egenskaper. Klicka på kryssrutan Exportera för ActionScript.
3. I fältet Klass anger du namnet som du vill använda när du refererar till det inbäddade ljudet i ActionScript. Som standard används ljudfilsnamnet i det här fältet. Om filnamnet innehåller en punkt som i namnet ”DrumSound.mp3” måste du ändra det till någonting liknande ”DrumSound”; det är inte tillåtet att använda punkter i klassnamn i ActionScript. I fältet Basklass ska det fortfarande stå flash.media.Sound.
4. Klicka på OK. Eventuellt visas en dialogruta som anger att ingen definition för basklassen hittades i klassökvägen. Klicka på OK och fortsätt. Om du anger ett klassnamn som inte överensstämmer med namnet i någon av klasserna i programmets klassökväg, genereras automatiskt en ny klass som ärver från flash.media.klassen Sound.
5. När du vill använda det inbäddade ljudet refererar du till klassnamnet för ljudet i ActionScript. Följande kod börjar med att skapa en ny instans av den automatiskt genererade klassen DrumSound:

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound är en underklass av flash.media.klassen Sound så den ärver klassen Sounds metoder och egenskaper inklusive `play()`-metoden som visas ovan.

Använda en inbäddad fil i Flex

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns flera olika sätt att bädda in ljud i ett Flex-program:

- Använda metadatataggen `[Embed]` i ett skript

- Använda direktivet `@Embed` i MXML för att tilldela en inbäddad resurs som en egenskap för en komponent som Button eller SoundEffect.
- Använda direktivet `@Embed` i en CSS-fil

I det här avsnittet beskrivs det första alternativet: att bädda in ljud i ActionScript-kod i ett Flex-program med hjälp av `[Embed]`-metadatataggen.

Om du vill bädda in en resurs i ActionScript-kod använder du metadatataggen `[Embed]`.

Placera ljudfilen i huvudkällmappen eller i en annan mapp som finns i projektets byggsökväg. När kompilatorn påträffar en Embed-metadatatagg skapas den inbäddade resursklassen. Du får tillgång till klassen via en variabel av datatypen Class som du deklarerar omedelbart efter `[Embed]`-metadatataggen.

I följande kod bäddas ett ljud med namnet smallSound.mp3 in och en variabel med namnet `soundClass` används för att lagra en referens till den inbäddade resursklassen som hör till ljudet. Koden skapar sedan en instans av den inbäddade resursklassen, byter ut den mot en instans av klassen Sound och anropar metoden `play()` för den instansen:

```
package
{
    import flash.display.Sprite;
    import flash.media.Sound;
    import flash.media.SoundChannel;

    public class EmbeddedSoundExample extends Sprite
    {
        [Embed(source="smallSound.mp3")]
        public var soundClass:Class;

        public function EmbeddedSoundExample()
        {
            var smallSound:Sound = new soundClass() as Sound;
            smallSound.play();
        }
    }
}
```

Om du vill använda det inbäddade ljudet för att ställa in en egenskap för en Flex-komponent, bör det bytas ut mot en instans av mx.core.klassen SoundAsset i stället för en instans av klassen Sound. Ett liknande exempel som använder klassen SoundAsset finns i ”Inbäddade resursklasser” i Learning ActionScript 3.0.

Arbeta med direktuppspelade ljudfiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När en ljudfil eller videofil spelas upp samtidigt som data fortfarande läses in kallas det för *direktuppspelning*. Externa ljudfiler som läses in från en fjärrserver är ofta direktuppspelade så att användaren inte behöver vänta på att alla ljuddata ska läsas in före avlyssning av ljudet.

Egenskapen `SoundMixer.bufferTime` representerar antalet millisekunder av ljuddata som Flash Player eller AIR ska samla in före ljudet spelas upp. M.a.o., om egenskapen `bufferTime` anges till 5000, läser Flash Player eller AIR in minst 5000 millisekunder av data från ljudfilen innan ljudet börjar spelas upp. Standardvärdet för `SoundMixer.bufferTime` är 1000.

Du kan ersätta det globala `SoundMixer.bufferTime`-värdet i ditt program för ett individuellt ljud genom att explicit specificera ett nytt `bufferTime`-värde när ljudet läses in. När du vill åsidosätta standardbufferttiden skapar du först en ny instans av klassen SoundLoaderContext och anger dess `bufferTime`-egenskap och skickar den sedan som en parameter till `Sound.load()`-metoden enligt exemplet nedan:

```
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();
```

Under uppspelningsförloppet försöker Flash Player och AIR behålla samma buffertstorlek eller större för ljudet. Om ljuddata läses in fortare än uppspelningshastigheten, fortsätter uppspelningen utan avbrott. Men om datainläsningshastigheten saktar ned på grund av nätverksbegränsningar kan spelhuvudet nå slutet på ljudbufferten. Om detta händer inaktiveras uppspelningen men återupptas automatiskt när mer ljuddata har lästs in.

Använd egenskapen `Sound.isBuffering` för att ta reda på om Flash Player eller AIR väntar på data för inläsning.

Arbeta med dynamiskt skapat ljud

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Obs! *Från och med Flash Player 10 och Adobe AIR 1.5 kan du generera ljud dynamiskt.*

I stället för att läsa in eller direktuppspela ett befintligt ljud, kan du skapa ljuddata dynamiskt. Du kan skapa ljuddata när du tilldelar en händelseavlyssnare för händelsen `sampleData` för ett Sound-objekt. (Händelsen `sampleData` definieras i klassen SampleDataEvent i flash.events-paketet.) I den här miljön läser Sound-objektet inte in ljuddata från någon fil. I stället fungerar det som en socket för ljuddata som börjar flöda med den funktion som du har tilldelat den här händelsen.

När du lägger till en `sampleData`-händelseavlyssnare till ett Sound-objekt begär objektet regelbundet data som ska läggas till i ljudbufferten. Den här bufferten innehåller data som Sound-objektet ska spela upp. När du anropar `play()`-metoden för Sound-objektet skickas händelsen `sampleData` när nya ljuddata begärs. (Detta gäller bara om Sound-objektet inte har läst in MP3-data från en fil.)

SampleDataEvent-objektet omfattar en `data`-egenskap. I händelseavlyssnaren ska du skriva ByteArray-objekt till det här `data`-objektet. De bytearrayer som du skriver till det här objektet lägger till buffertlagrade ljuddata som Sound-objektet spelar upp. Bytearrayen i bufferten är en ström med flyttalsvärden från -1 till 1. Varje flyttalsvärde representerar amplituden för en kanal (vänster eller höger) för en ljudsampling. Ljud samplas med 44 100 samplingar per sekund. Varje sampling innehåller en höger- och vänsterkanal, sammanflätade som flyttalsdata i bytearrayen.

I hanterarfunktionen använder du metoden `ByteArray.writeFloat()` för att skriva till `data`-egenskapen för händelsen `sampleData`. Följande kod genererar till exempel en sinusvåg:

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

När du anropar `Sound.play()` börjar programmet att anropa händelsehanteraren, och begär ljudsamplingsdata. Programmet fortsätter att skicka händelser när ljudet spelas upp fram till dess att du inte anger mer data eller tills `SoundChannel.stop()` anropas.

Händelsens tidsfördröjning kan variera från plattform till plattform och kan ändras i framtida versioner av Flash Player och AIR. Var inte beroende av en specifik tidsfördröjning, beräkna den i stället. Tidsfördröjningen beräknas med följande formel:

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Ange 2 048 till och med 8 192 samplingar för egenskapen `data` för SampleDataEvent-objektet (för varje anrop till händelseavlyssnaren). Ju fler samplingar (upp till 8 192), desto bättre prestanda. Ju färre samplingar, desto mer troligt är det att störningar inträffar under uppspelning. Det här beteendet är olika på olika plattformar och inträffar i olika situationer, till exempel när webbläsare ändrar storlek. Kod som fungerar på en plattform när du bara anger 2 048 samplingar kanske inte fungerar när den körs på en annan plattform. Om du kräver lägsta möjliga tidsfördröjning bör du överväga att göra mängden data valbar för användaren.

Om du anger färre än 2 048 samplingar (per anrop till händelseavlyssnaren `sampleData`) stannar programmet när resten av samplingarna har spelats upp. SoundChannel-objektet skickar sedan en SoundComplete-händelse.

Ändra ljud från MP3-data

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Använd metoden `Sound.extract()` när du vill hämta data från ett Sound-objekt. Du kan använda (och ändra) dessa data om du vill skriva till den dynamiska strömmen för ett annat Sound-objekt för uppspelning. Följande kod använder byte för en inläst MP3-fil och skickar dem via en filterfunktion, `upOctave()`:

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

Begränsningar för genererade ljud

Flash Player 10 och senare, Adobe AIR 1.5 och senare

När du använder en `sampleData`-händelseavlyssnare med ett Sound-objekt är de enda övriga Sound-metoder som aktiveras `Sound.extract()` och `Sound.play()`. Anrop av andra metoder eller egenskaper resulterar i ett undantag. Alla metoder och egenskaper för SoundChannel-objektet är fortfarande aktiverade.

Spela upp ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att spela upp ett inläst ljud kan vara så enkelt som att anropa metoden `Sound.play()` för ett Sound-objekt, enligt nedanstående:

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

När ljud spelas upp med ActionScript 3.0 kan du utföra följande:

- Spela upp ett ljud från en viss startposition

- Pausa ett ljud och återuppta uppspelningen från samma position senare
- Veta exakt när en ljuduppspelning slutar
- Spåra uppspelningsförloppet för ett ljud.
- Ändra volym och panorering medan ett ljud spelas upp

Använd klasserna SoundChannel, SoundMixer och SoundTransform när du vill utföra de här åtgärderna under uppspelning.

Klassen SoundChannel styr uppspelningen av ett enstaka ljud. Du kan tänka på egenskapen `SoundChannel.position` som ett spelhuvud som indikerar den aktuella ljuddatapositionen som just nu spelas upp.

När programmet anropar en `Sound.play()`-metod skapas en ny instans av klassen SoundChannel för styrning av uppspelningen.

Programmet kan spela upp ett ljud från en specifik startposition genom att överföra positionen, uttryckt i millisekunder, som `startTime`-parameter för `Sound.play()`-metoden. Det kan också specificera ett fast antal gånger som ljudet ska repeteras i snabb följd genom att överföra ett numeriskt värde i parametern `loops` för `Sound.play()`-metoden.

När `Sound.play()`-metoden anropas med både en `startTime`-parameter och en `loops`-parameter, spelas ljudet upp upprepade gånger från samma referenspunkt varje gång enligt exemplet i följande kod:

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

I det här exemplet spelas ljudet upp från en punkt en sekund efter att ljudet har startats och tre gånger i följd.

Pausa och fortsätta ljuduppspelning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om långa ljud spelas upp i programmet, som låtar eller poddsändningar, vill du antagligen att användarna ska kunna pausa och återuppta uppspelningen av dessa ljud. Ett ljud kan egentligen inte pausas under uppspelning i ActionScript; det kan endast stoppas. Men ett ljud kan spelas upp med start från en viss punkt. Du kan spela in ljudpositionen när den stoppas och sedan spela upp ljudet med start från den positionen senare.

Vi antar att koden läser in och spelar en ljudfil liknande den här:

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

När ljudet spelas upp indikerar `SoundChannel.position`-egenskapen den position i ljudfilen som spelas upp just nu. Programmet kan lagra positionsvärdet innan ljudet stoppas:

```
var pausePosition:int = channel.position;
channel.stop();
```

För att återuppta uppspelningen av ljudet överför du det tidigare lagrade positionsvärdet för att återstarta ljudet från samma position det stoppade på förut.

```
channel = snd.play(pausePosition);
```

Övervaka uppspelning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet behöver en indikation om när en ljuduppspelning stoppas så det kan starta uppspelningen av ett annat ljud eller rensa några resurser som använts under den tidigare uppspelningen. Klassen SoundChannel skickar en `Event.SOUND_COMPLETE`-händelse när dess ljuduppspelning är slut. Programmet kan lyssna efter händelsen och utföra lämplig åtgärd enligt exemplet nedan:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

Klassen SoundChannel skickar inte progress-händelser under uppspelning. För rapport om uppspelningsförloppet kan du ange att programmet ska ha en egen timingmekanism och spåra positionen för spelhuvudet.

För beräkning av hur stor procentandel av ljudet som har spelats upp kan du dividera värdet för `SoundChannel.position`-egenskapen med längden för ljuddata som spelas upp:

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Men den här koden rapporterar endast rätt procentsats för uppspelning om ljuddata var fullständigt inläst innan uppspelningen började. Egenskapen `Sound.length` visar storleken på ljuddata som för närvarande är inläst, inte den slutliga storleken för hela ljudfilen. Om du vill spåra uppspelningsförloppet för ett direktuppspelat ljud som fortfarande läses in, kan programmet beräkna den slutliga storleken för hela ljudfilen och använda det värdet vid beräkningarna. Du kan beräkna den slutliga längden för ljuddata med hjälp av egenskaperna `bytesLoaded` och `bytesTotal` för Sound-objektet enligt följande:

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

I följande kod läses en större ljudfil in och `Event.ENTER_FRAME`-händelsen används som timingmekanism för att visa uppspelningsförloppet. Periodvis rapporteras hur stor procent som spelats upp, vilket beräknas som nuvarande positionsvärde dividerat med den totala längden av ljuddata:

```
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);
channel.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);
}
```

När ljuddata börjar läsas in anropar koden metoden `snd.play()` och lagrar det slutliga SoundChannel-objektet i variabeln `channel`. Sedan läggs en händelseavlyssnare till i huvudprogrammet för `Event.ENTER_FRAME`-händelsen och en annan händelseavlyssnare till SoundChannel-objektet för `Event.SOUND_COMPLETE`-händelsen som inträffar när uppspelningen är slutförd.

Varje gång programmet kommer till en ny bildruta i animeringen anropas `onEnterFrame()`-metoden. Metoden `onEnterFrame()` beräknar den totala längden för ljudfilen baserat på den datamängd som redan har lästs in och sedan beräknar den och visar hur stor procent som spelats upp.

När hela ljudet har spelats upp, körs `onPlaybackComplete()`-metoden som tar bort händelseavlyssnaren från `Event.ENTER_FRAME`-händelsen så att den inte försöker visa förloppsuppdateringar när uppspelningen är klar.

Händelsen `Event.ENTER_FRAME` kan skickas många gånger per sekund. I vissa fall vill du inte visa uppspelningsförloppet så ofta. I dessa fall kan du ange en egen timingmekanism med klassen flash.util.Timer; se ”Arbeta med datum och tidpunkter” på sidan 1.

Stoppa direktuppspelade ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns en egenhet i uppspelningsprocessen för ljud som direktuppspelas, alltså för ljud som fortfarande läses in medan de spelas upp. När programmet anropar metoden `SoundChannel.stop()` i en SoundChannel-instans som spelar upp ett direktuppspelat ljud, stoppas ljuduppspelningen först i en bildruta och sedan, i nästa bildruta, startas det om från början av ljudet. Det här sker för att ljudinläsningsförloppet fortfarande pågår. Om du vill stoppa både inläsningen och uppspelningen av ett direktuppspelat ljud anropar du `Sound.close()`-metoden.

Säkerhetsaspekter vid inläsning och uppspelning av ljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmets möjligheter att nå ljuddata kan vara begränsade enligt säkerhetsmodellen i Flash Player eller AIR. Varje ljud begränsas av två olika säkerhetssandlådor, sandlådan för innehållet (”innehållssandlådan”) och sandlådan för programmet eller objektet som läses in och spelar upp ljudet (”ägarsandlådan”). När det gäller AIR-programinnehåll i säkerhetssandlådan application, är alla ljud (även dem som läses in från andra domäner) tillgängliga för innehållet i säkerhetssandlådan application. Innehåll i andra säkerhetssandlådor följer emellertid samma regler som innehåll som körs i Flash Player. Mer information om säkerhetsmodellen i Flash Player i allmänhet, och definitionen av sandlåda finns i avsnittet ”Säkerhet” på sidan 1047.

Innehållssandlådan styr om detaljerad ljuddata kan extraheras från ljudet med `id3`-egenskapen eller `SoundMixer.computeSpectrum()`-metoden. Den begränsar inte inläsningen eller uppspelningen av själva ljudfilen.

Ursprungsdomänen för ljudfilen definierar säkerhetsbegränsningarna för innehållssandlådan. Vanligtvis har programmet eller objektet fullständig åtkomst till ljudfilen om en den finns i samma domän eller mapp som SWF-filen för programmet eller objektet som läser in den. Om ljudet kommer från en annan domän än programmet kan det fortfarande läsas in till innehållssandlådan via en principfil.

Programmet kan överföra ett SoundLoaderContext-objekt med en `checkPolicyFile`-egenskap som en parameter till `Sound.load()`-metoden. Om du sätter egenskapen `checkPolicyFile` till `true` söker Flash Player eller AIR efter en principfil på servern som ljudet läses in från. Om det finns en principfil och den godkänner domänen för den inläsande SWF-filen, kan SWF-filen läsa in ljudfilen, få åtkomst till `id3`-egenskapen i Sound-objektet och anropa `SoundMixer.computeSpectrum()`-metoden för inlästa ljud.

Ägarsandlådan styr lokal uppspelning av ljuden. Programmet eller objektet som startar uppspelningen av ett ljud definieras som ägarsandlådan.

Metoden `SoundMixer.stopAll()` avbryter ljuden i alla SoundChannel-objekt som spelas upp om de uppfyller följande kriterier:

- Ljuden startades av objekt inom samma ägarsandlåda.
- Ljuden kommer från en källa med en principfil som godkänner åtkomst till domänen för programmet eller objektet som anropar `SoundMixer.stopAll()`-metoden.

I ett AIR-program begränsas emellertid inte innehåll i säkerhetssandlådan application (innehåll som installeras med AIR-programmet) av de här säkerhetsbegränsningarna.

För att ta reda på om `SoundMixer.stopAll()`-metoden faktiskt kommer att stoppa alla ljud som spelas upp kan programmet anropa metoden `SoundMixer.areSoundsInaccessible()`. Om metoden returnerar värdet `true` omfattas några av ljuden som spelas upp inte av styrningen från den aktuella ägarsandlådan och kommer inte att stoppas av `SoundMixer.stopAll()`-metoden.

Metoden `SoundMixer.stopAll()` stoppar också spelhuvudet från uppspelning av ljud som lästes in från externa filer. Däremot kan ljud som är inbäddade i FLA-filer och kopplade till bildrutor på tidslinjen som använder Flash-utvecklingsverktyget börja spelas upp igen om animeringen flyttas till en ny bildruta.

Styra ljudvolym och panorering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett individuellt SoundChannel-objekt styr både vänster och höger stereokanal för ett ljud. Om ett MP3-ljud är mono, innehåller de vänstra och högra stereokanalerna för SoundChannel-objektet identiska ljudvågor.

Du kan ta reda på amplituden i varje stereokanal för ljudet som spelas upp med `leftPeak`- och `rightPeak`-egenskaperna i SoundChannel-objektet. De här egenskaperna visar maximal amplitud för själva ljudvågen. De representerar inte den faktiska uppspelningsvolymen. Den faktiska uppspelningsvolymen är en funktion för amplituden för ljudvågen och volymvärdena som angetts i SoundChannel-objektet och klassen SoundMixer.

Du kan använda egenskapen pan för ett SoundChannel-objekt för att specificera olika volymnivåer för var och en av de vänstra och högra kanalerna under uppspelningen. Pan-egenskapen kan ha ett värde från -1 till 1, där -1 betyder att den vänstra kanalen spelar på högsta volym medan den högra kanalen är tyst och 1 betyder att den högra kanalen spelar på högsta volym medan den vänstra kanalen är tyst. Numeriska värden mellan -1 och 1 anger proportionella värden för de vänstra och högra kanalvärdena och värdet 0 betyder att båda kanalerna spelar på en balanserad, medelvolymnivå.

I följande kodexempel skapas ett SoundTransform-objekt med ett volymvärde på 0,6 och ett panoreringsvärde på -1 (högsta volym på vänstra kanalen och ingen volym på den högra kanalen). SoundTransform-objektet överförs som en parameter till `play()`-metoden vilken använder SoundTransform-objektet på det nya SoundChannel-objektet som skapas för att styra uppspelningen.

```
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);
```

Du kan ändra volym och panorering medan ett ljud spelas upp genom att ange `pan`- eller `volume`-egenskaperna för ett SoundTransform-objekt och sedan använda objektet som `soundTransform`-egenskap för ett SoundChannel-objekt.

Du kan också ange globala volym- och panoreringsvärden för alla ljud på en gång med `soundTransform`-egenskapen för klassen SoundMixer enligt följande exempel:

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Du kan också använda ett SoundTransform-objekt för att ange volym- och panoreringsvärden för ett Microphone-objekt (se ”Hämta ljudindata” på sidan 461) och för Sprite-objekt och SimpleButton-objekt.

I följande exempel växlar panoreringen av ljudet från den vänstra kanalen till den högra och tillbaka igen medan ljudet spelas upp.

```
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; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Koden börjar med att läsa in en ljudfil och sedan skapa ett nytt SoundTransform-objekt med volymen inställd på 1 (full volym) och panoreringen på 0 (jämnt balanserat mellan vänster och höger). Sedan anropas metoden `snd.play()` och SoundTransform-objektet skickas som en parameter.

När ljudet spelas upp körs `onEnterFrame()`-metoden upprepade gånger. `onEnterFrame()`-metoden använder `Math.sin()`-funktionen för generering av ett värde mellan -1 och 1, ett intervall som motsvarar de acceptabla värdena för `SoundTransform.pan`-egenskapen. Det nya värdet anges för `pan`-egenskaper i SoundTransform-objektet och kanalens `soundTransform`-egenskap ställs in att använda det ändrade SoundTransform-objektet.

Om du vill köra det här exemplet byter du ut filnamnet bigSound.mp3 mot en lokal MP3-fil. Sedan kör du exemplet. Nu hör du hur den vänstra kanalvolymen höjs medan den högra kanalvolymen sänks och omvänt.

I det här exemplet kan samma effekt uppnås genom att ange `soundTransform`-egenskapen för klassen SoundMixer. Men det skulle påverka panoreringen av alla ljud som spelas upp, inte endast ljudet som spelas upp av det här SoundChannel-objektet.

Arbeta med ljudmetadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ljudfiler som använder MP3-format kan innehålla ytterligare data om ljudet i form av ID3-taggar.

Alla MP3-filer innehåller inte ID3-metadata. När ett Sound-objekt läser in en MP3-ljudfil, skickas en `Event.ID3`-händelse om ljudfilen innehåller ID3-metadata. För att undvika körningsfel, ska programmet vänta på mottagning av `Event.ID3`-händelsen innan `Sound.id3`-egenskapen för ett inläst ljud hämtas.

I följande kod visas hur programmet känner igen när ID3-metadata för en ljudfil har lästs in:

```
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]);
    }
}
```

Koden börjar med att skapa ett Sound-objekt och ange att avlyssning av en `Event.ID3`-händelse ska ske. När ljudfilens ID3-metadata har lästs in anropas metoden `onID3InfoReceived()`. Målet för Event-objektet som överförs till `onID3InfoReceived()`-metoden är det ursprungliga Sound-objektet, så metoden får sedan Sound-objektets `id3`-egenskap och itererar igenom alla sina namngivna egenskaper för att spåra deras värden.

Använda råa ljuddata

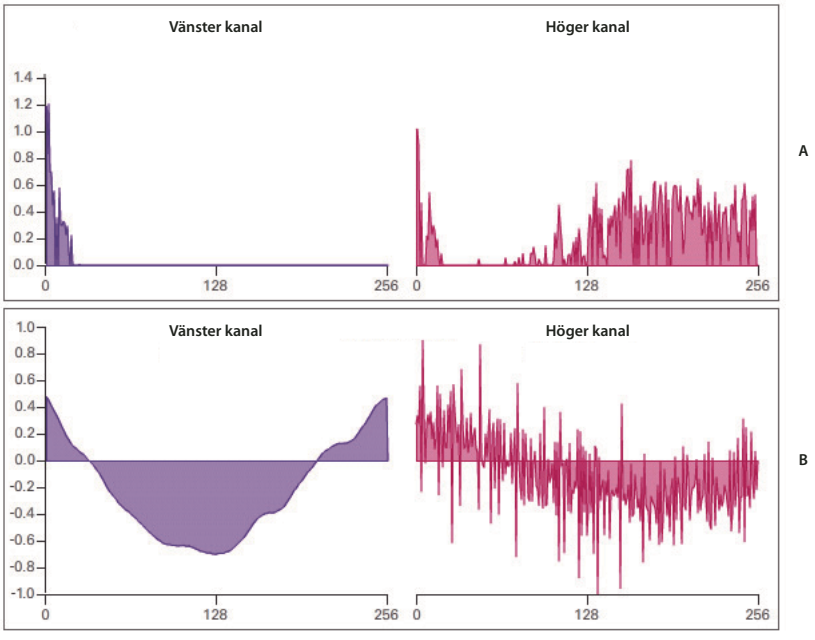
Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `SoundMixer.computeSpectrum()` kan ett program läsa råa ljuddata för den vågform som spelas upp. Om mer än ett SoundChannel-objekt spelas upp visar `SoundMixer.computeSpectrum()`-metoden kombinerat ljuddata för alla SoundChannel-objekt tillsammans.

Ljuddata returneras som ett ByteArray-objekt på 512 byte, där vart och ett innehåller ett flyttalsvärde mellan -1 och 1. De här värdena representerar amplituden för positionerna i ljudvågformen som spelas upp. Värdena levereras i två grupper om 256, den första gruppen för den vänstra stereokanalen och den andra gruppen för den högra stereokanalen.

`SoundMixer.computeSpectrum()`-metoden returnerar frekvensspektrumdata i stället för vågformsdata om parametern `FFTMode` har värdet `true`. I frekvensspektrumet visas amplituden arrangerad per ljudfrekvens från den lägsta frekvensen till den högsta. Fast Fourier Transform (FFT) används för att konvertera vågformsdata till frekvensspektrumdata. Det resulterande frekvensspektrumvärdet ligger i ett intervall från 0 till ungefär 1,414 (kvadratroten ur 2).

I följande diagram jämförs data som returnerats från `computeSpectrum()`-metoden när `FFTMode`-parametern angetts till `true` och när den har angetts till `false`. Ljuddata till det här diagrammet innehåller ett högt basljud i den vänstra kanalen och ett trumsolo i den högra kanalen.



Värden som returnerats av metoden SoundMixer.computeSpectrum()
A. *fftMode = true* ***B.*** *fftMode = false*

`computeSpectrum()` kan också returnera data som har samplats om på en lägre bithastighet. Vanligtvis resulterar det här i jämnare vågformsdata eller frekvensdata men det försämrar detaljerna. Parametern `stretchFactor` styr hastigheten med vilken `computeSpectrum()`-metoddata samplas. När parametern `stretchFactor` har värdet 0 (standardvärdet) samplas ljuddata med en hastighet på 44.1 kHz. Hastigheten halveras för varje värde i stretchFactor-parametern så värdet 1 specificerar en hastighet av 22.05 kHz, värdet 2 specificerar en hastighet av 11.025 kHz o.s.v. Metoden `computeSpectrum()` returnerar fortfarande 256 byte per stereokanal när ett högre `stretchFactor`-värde används.

`SoundMixer.computeSpectrum()`-metoden har några begränsningar:

- Eftersom ljuddata från en mikrofon eller från RTMP-direktuppspelning inte överförs via det globala SoundMixer-objektet kommer `SoundMixer.computeSpectrum()`-metoden inte att returnera data från dessa källor.
- Om fler än ett av ljuden som spelas upp kommer från källor utanför den aktuella innehållssandlådan, kommer säkerhetsbegränsningarna att få `SoundMixer.computeSpectrum()`-metoden att orsaka ett fel. Mer information om säkerhetsbegränsningarna i metoden `SoundMixer.computeSpectrum()` finns i ”Säkerhetsaspekter vid inläsning och uppspelning av ljud” på sidan 454 och ”Åtkomst av inlästa medier som data” på sidan 1070.

I ett AIR-program begränsas emellertid inte innehåll i säkerhetssandlådan application (innehåll som installeras med AIR-programmet) av de här säkerhetsbegränsningarna.

Skapa en enkel ljudvisualisering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

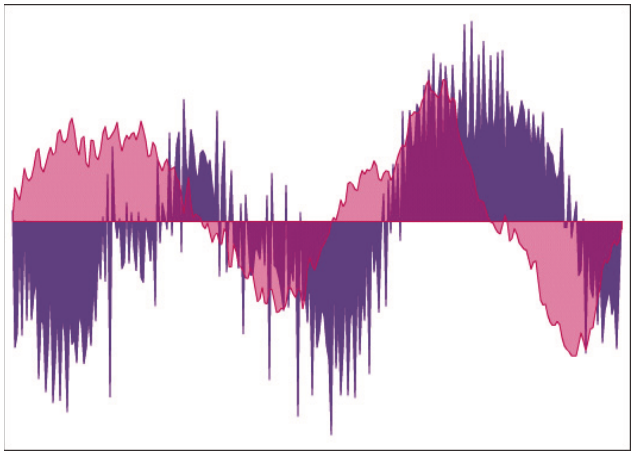
I följande exempel används `SoundMixer.computeSpectrum()`-metoden för att visa ett diagram över ljudvågformen som animeras med varje bildruta:

```
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;

    // left channel
    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();

    // right channel
    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);
}
```

Först läses en ljudfil in och spelas upp och sedan avlyssnas `Event.ENTER_FRAME`-händelsen som aktiverar `onEnterFrame()`-metoden medan ljudet spelas upp. Metoden `onEnterFrame()` börjar med att anropa `SoundMixer.computeSpectrum()`-metoden vilken lagrar ljudvågsdata i ByteArray-objektet `bytes`.

Ljudvågsformen ritas med vektorritnings-API. En `for`-slinga går igenom de första 256 datavärdena som representerar den vänstra stereokanalen och ritar en linje från punkt till punkt med `Graphics.lineTo()`-metoden. En andra `for`-slinga går igenom de följande 256 värdena och ritar dem i motsatt ordning den här gången, från höger till vänster. De slutliga vågformsritningarna kan ge en intressant spegelbildseffekt enligt exempel i följande bild.



Hämta ljudindata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen Microphone kan programmet ansluta till en mikrofon eller annan ljudindataenhet i användarens system och sända indataljudet till högtalarna i det systemet eller skicka ljuddata till en fjärrserver, t.ex. Flash Media Server. Du kan komma åt de råa ljuddata från mikrofonen och spela in eller bearbeta dem. Du kan också skicka ljudet direkt till systemets högtalare eller skicka komprimerade ljuddata till en fjärrserver. Du kan använda Speex- eller Nellymoser-kodeken för data som skickas till en fjärrserver. (Speex-kodeken stöds från och med Flash Player 10 och Adobe AIR 1.5.)

Fler hjälpavsnitt

Michael Chaize: AIR, Android, and the Microphone (AIR, Android och mikrofoner)

Christophe Coenraets: VoiceNotes for Android (VoiceNotes för Android)

Använda en mikrofon

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Microphone har ingen konstruktormetod. I stället använder du den statiska `Microphone.getMicrophone()`-metoden för att hämta en ny Microphone-instans enligt nedan:

```
var mic:Microphone = Microphone.getMicrophone();
```

Om du anropar `Microphone.getMicrophone()`-metoden utan en parameter, returneras den första ljudindataenheten som påträffas i användarens system.

Ett system kan ha fler än en ljudindataenhet ansluten. Programmet kan använda egenskapen `Microphone.names` för att få en array med namnen på alla tillgängliga ljudindataenheter. Sedan kan det anropa `Microphone.getMicrophone()`-metoden med en `index`-parameter som matchar indexvärdet för enhetens namn i arrayen.

Ett system kanske inte har någon mikrofon eller andra indataenheter för ljud anslutna. Du kan använda egenskapen `Microphone.names` eller metoden `Microphone.getMicrophone()` för att kontrollera om användaren har någon ljudindataenhet installerad. Om användaren inte har installerat någon ljudindataenhet har `names`-arrayen längden noll och `getMicrophone()`-metoden returnerar värdet `null`.

När programmet anropar `Microphone.getMicrophone()`-metoden, visar Flash Player dialogrutan Flash Player-inställningar som uppmanar användaren att antingen tillåta eller neka Flash Player åtkomst till kameran och mikrofonen i systemet. När användaren har klickat på Tillåt eller Neka i dialogrutan skickas ett StatusEvent-objekt. Egenskapen `code` i StatusEvent-instansen anger om åtkomst till mikrofonen tilläts eller nekades enligt det här exemplet:

```
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`-egenskapen innehåller ”Microphone.Unmuted” om åtkomst tilläts eller ”Microphone.Muted” om åtkomst nekades.

Egenskapen `Microphone.muted` anges till `true` respektive `false` när användaren tillåter eller nekar åtkomst till mikrofonen. Men egenskapen `muted` anges inte för Microphone-instansen förrän StatusEvent-händelsen har skickats, och därför bör programmet också vänta tills händelsen `StatusEvent.STATUS` skickas innan `Microphone.muted`-egenskapen kontrolleras.

För att Flash Player ska kunna visa dialogrutan med inställningar måste programfönstret vara tillräckligt stort (minst 215 x 138 pixlar). Annars nekas åtkomst automatiskt.

Innehåll som körs i AIR-programmets sandlåda behöver inte användarens tillåtelse för att komma åt mikrofonen. Därför skickas aldrig några statushändelser för ljud av och ljud på för mikrofonen. Innehåll som körs i AIR utanför programmets sandlåda behöver däremot användarens tillåtelse så dessa statushändelser kan skickas.

Dirigera mikrofonljud till lokala högtalare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ljudindata från en mikrofon kan dirigeras till de lokala högtalarna genom anrop av `Microphone.setLoopback()`-metoden med parametervärdet `true`.

När ljud från en lokal mikrofon dirigeras till lokala högtalare finns det risk att rundgång skapas, vilket kan förorsaka högt och skrikigt ljud och kan leda till att ljudhårdvaran kan skadas. Genom att anropa metoden `Microphone.setUseEchoSuppression()` med parametervärdet `true` reduceras risken för rundgång, men den elimineras inte helt. Adobe rekommenderar att du alltid anropar `Microphone.setUseEchoSuppression(true)` före anropet av `Microphone.setLoopback(true)` såvida du inte är säker på att användarna spelar upp ljudet i hörlurar eller liknande, men inte i högtalare.

I följande kod visas hur ljud dirigeras från en lokal mikrofon till de lokala högtalarna:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Ändra mikrofonljud

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet kan ändra ljuddata som kommer från en mikrofon på två sätt. Det kan ändra förstärkningen av indataljudet, vilket i praktiken innebär att indatavärden multipliceras med ett specificerat tal för att skapa ett högre eller lägre ljud. Egenskapen `Microphone.gain` accepterar numeriska värden från 0 t.o.m. 100. Värdet 50 fungerar som en multiplikator på ett och anger normal volym. Värdet noll fungerar som en multiplikator på noll och avbryter indataljudet. Värden över 50 anger högre volymer än normalt.

Programmet kan också ändra samplingsfrekvensen för indataljud. Högre samplingsfrekvenser ökar ljudkvaliteten men de skapar också tätare dataflöden som använder mer resurser för överföring och lagring. Egenskapen `Microphone.rate` representerar ljudets samplingsfrekvens mätt i kilohertz (kHz). Standardsamplingsfrekvensen är 8 kHz. Du kan ange ett högre värde än 8 kHz för `Microphone.rate`-egenskapen om mikrofonen klarar den högre frekvensen. Om du t.ex. anger värdet 11 för `Microphone.rate`-egenskapen, blir samplingsfrekvensen 11 kHz. Värdet 22 ger samplingsfrekvensen 22 kHz och så vidare. Vilka samplingsfrekvenser som finns beror på den valda koden. När du använder Nellymoser-koden kan du ange 5, 8, 11, 16, 22 och 44 kHz som samplingsfrekvens. När du använder Speex-kodeken (som är tillgänglig från och med Flash Player 10 och Adobe AIR 1.5) kan du bara använda 16 kHz.

Identifiera mikrofonaktivitet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att spara bandbredd och bearbetningsresurser försöker Flash Player identifiera när inget ljud överförs via en mikrofon. När mikrofonens aktivitetsnivå stannar under tystnadsnivåtröskeln för en tidsperiod, slutar Flash Player att överföra ljudindata och skickar en enkel ActivityEvent i stället. När du använder Speex-kodeken (som är tillgänglig från och med Flash Player 10 eller senare och Adobe AIR 1.5 eller senare) ställer du in tystnadsnivån på 0 för att försäkra att ljudinformationen överförs kontinuerligt. Speex-röstidentifiering minskar bandbredden automatiskt.

Obs! *Ett Microphone-objekt skickar bara aktivitetshändelser när programmet övervakar mikrofonen. Därför skickas inga aktivitetshändelser om du inte anropar `setLoopBack( true )`, lägger till en lyssnare för samplingsdatahändelser eller kopplar mikrofonen till ett NetStream-objekt.*

Tre egenskaper i klassen Microphone övervakar och styr identifieringen av aktivitet:

- Den skrivskyddade `activityLevel`-egenskapen indikerar mängden ljud som mikrofonen identifierar på en skala från 0 till 100.
- Med `silenceLevel`-egenskapen specificeras mängden ljud som behövs för att aktivera mikrofonen och skicka en `ActivityEvent.ACTIVITY`-händelse. Egenskapen `silenceLevel` använder också en skala från 0 till 100, och standardvärdet är 10.
- Egenskapen `silenceTimeout` beskriver antal millisekunder som aktivitetsnivån måste vara under tystnadsnivån tills en `ActivityEvent.ACTIVITY`-händelse skickas för att indikera att mikrofonen nu är tyst. Standardvärdet för `silenceTimeout` är 2000.

Båda egenskaperna `Microphone.silenceLevel` och `Microphone.silenceTimeout` är skrivskyddade men deras värden kan ändras med `Microphone.setSilenceLevel()`-metoden.

I några fall kan proceduren för aktivering av mikrofonen förorsaka en kort fördröjning när ny aktivitet identifieras. Om mikrofonen alltid är aktiverad kan sådan aktiveringsfördröjning tas bort. Programmet kan anropa `Microphone.setSilenceLevel()`-metoden med värdet noll i `silenceLevel`-parametern som indikerar för Flash Player att hålla mikrofonen aktiv och fortsätta samla ljuddata även om inget ljud identifieras. Omvänt förhindras mikrofonen från att aktiveras om värdet 100 anges i `silenceLevel`-parametern.

Följande exempel visar information om mikrofonen och rapporterar om aktivitets- och statushändelser som skickas av ett Microphone-objekt:

```
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);
}
```

När du kör exemplet ovan, pratar eller testar ljud i systemmikrofonen och tittar på de resulterande trace-programsatserna som visas på en konsol eller i ett felsökningsfönster.

Skicka ljud till och från en medieserver

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ytterligare ljudmöjligheter finns tillgängliga när ActionScript används med direktuppspelning med en mediaserver som Flash Media Server (FMS).

Särskilt kan programmet bifoga ett Microphone-objekt till ett NetStream-objekt och överföra data direkt från användarens mikrofon till servern. Ljuddata kan också direktuppspelas från servern till ett program och spelas upp som en del av ett MovieClip-objekt eller via ett Video-objekt.

Speex-kodeken är tillgänglig från och med Flash Player 10 och Adobe AIR 1.5. Om du vill ställa in kodeken som används för komprimerat ljud som skickas till medieservern ställer du in `codec`-egenskapen för Microphone-objektet. Den här egenskapen kan ha två värden, som räknas upp i klassen SoundCodec. Om du ställer in kodegenskapen på `SoundCodec.SPEEX` väljs Speex-kod för ljudkomprimering. Om du ställer in egenskapen på `SoundCodec.NELLYMOSER` (standardvärde) väljs Nellymoser-koden för ljudkomprimering.

Mer information finns i dokumentationen till Flash Media Server på www.adobe.com/go/learn_fms_docs_se.

Hämta mikrofonljuddata

Flash Player 10.1 och senare, Adobe AIR 2 och senare

I Flash Player 10.1 och AIR 2 (eller senare versioner) kan du hämta data från en mikrofon som en bytearray med flyttalsvärden. Varje värde motsvarar en sampling med monofoniska ljuddata.

Om du vill hämta mikrofondata anger du en händelseavlyssnare för händelsen `sampleData` i Microphone-objektet. Microphone-objektet skickar `sampleData`-händelser med jämna mellanrum när mikrofonbufferten fylls med ljudsamplingar. SampleDataEvent-objektet har en `data`-egenskap som är en bytearray med ljud samplingar. Alla samplingar representeras som flyttalsvärden, och alla värden motsvarar en monofonisk ljudsampling.

Följande kod hämtar mikrofonljuddata till ett ByteArray-objekt med namnet `soundBytes`:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
function micSampleDataHandler(event:SampleDataEvent):void {
    while(event.data.bytesAvailable) {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
```

Du kan återanvända samplingsbyte som uppspelningsljud för ett Sound-objekt. Om du gör det ska du ange egenskapen `rate` för Microphone-objektet till 44, vilket är den samplingsfrekvens som används av Sound-objekt. (Du kan också konvertera mikrofonsamplingar som hämtats med en lägre frekvens till frekvensen 44 kHz, som krävs för Sound-objektet.) Tänk också på att Microphone-objektet hämtar monofoniska samplingar, men att Sound-objektet använder stereoljud. Därför får du skriva de byte som hämtats av Microphone-objektet till Sound-objektet två gånger. I följande exempel hämtas 4 sekunder med mikrofondata och spelas sedan upp med hjälp av ett Sound-objekt:

```
const DELAY_LENGTH:int = 4000;
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.gain = 100;
mic.rate = 44;
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);

var timer:Timer = new Timer(DELAY_LENGTH);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function micSampleDataHandler(event:SampleDataEvent):void
{
    while(event.data.bytesAvailable)
    {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
var sound:Sound = new Sound();
var channel:SoundChannel;
function timerHandler(event:TimerEvent):void
{
    mic.removeEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
    timer.stop();
    soundBytes.position = 0;
    sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playbackSampleHandler);
    channel.addEventListener( Event.SOUND_COMPLETE, playbackComplete );
    channel = sound.play();
}

function playbackSampleHandler(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192 && soundBytes.bytesAvailable > 0; i++)
    {
        trace(sample);
        var sample:Number = soundBytes.readFloat();
        event.data.writeFloat(sample);
        event.data.writeFloat(sample);
    }
}

function playbackComplete( event:Event ):void
{
    trace( "Playback finished.");
}
```

Mer information om att spela upp ljud från ljudsamplingsdata finns i ”Arbeta med dynamiskt skapat ljud” på sidan 448.

Exempel på ljud: Podcast Player

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En poddsändning är en ljudfil som distribueras över Internet på begäran eller genom prenumeration. Poddsändningar är vanligtvis publicerade som en del av en serie vilket också kallas en poddsändningskanal. Eftersom poddsändningsavsnitt kan vara allt från en minut till många timmar långa är de vanligtvis direktuppspelade. Poddsändningsavsnitt som också kallas objekt levereras vanligtvis i filformatet MP3. Videopoddsändningar är också populära men i det här exemplet spelas endast ljudpoddsändningar upp som använder MP3-filer.

Det här exemplet är inte ett aggregeringsprogram för poddsändning med alla funktioner. Det hanterar t.ex. inte prenumerationer på specifika poddsändningar och kommer inte ihåg vilka poddsändningar användaren har lyssnat på nästa gång programmet körs. Det kan fungera som en referenspunkt för en poddsändningsaggregering med fler funktioner.

Podcast Player-exemplet illustrerar följande tekniker för ActionScript-programmering:

- Läsa en extern RSS-matning och tolka dess XML-innehåll
- Skapa en SoundFacade-klass för att förenkla inläsning och uppspelning av ljudfiler
- Visa ljuduppspelningsförlopp
- Pausa och återuppta uppspelning av ljud

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Podcast Player-programfilerna finns i mappen Samples/PodcastPlayer. Programmet består av följande filer:

Fil	Beskrivning
PodcastPlayer.mxml eller PodcastPlayer.fla	Användargränssnittet för programmet för Flex (MXML) eller Flash (FLA).
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	Dokumentklass som innehåller användargränssnittslogiken för Podcast Player (endast Flash).
SoundPlayer.mxml	En MXML-komponent som visar uppspelningsknappar och förloppsindikatorer, och styr uppspelning av ljud. Gäller enbart för Flex.
main.css	Format för programmets användargränssnitt (endast Flex).
bilder/	Ikoner för att formatera knapparna (endast Flex).
comp/example/programmingas3/podcastplayer/SoundPlayer.as	Klass för SoundPlayer-filmklippssymbolen som innehåller användargränssnittslogiken för ljudspelaren (endast Flash).
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	Anpassad cellrenderare för att visa en Play-knapp i en datarutnätscell (endast Flash).
com/example/programmingas3/podcastplayer/RSSBase.as	En basklass med allmänna egenskaper och metoder för klassen RSSChannel och klassen RSSItem.
com/example/programmingas3/podcastplayer/RSSChannel.as	En ActionScript-klass som innehåller data om en RSS-kanal.

Fil	Beskrivning
com/example/programmingas3/podcastplayer/RSSItem.as	En ActionScript-klass som innehåller data om ett RSS-objekt.
com/example/programmingas3/podcastplayer/SoundFacade.as	Huvudklassen för ActionScript i programmet. Den kapslar in metoderna och händelserna i klassen Sound och klassen SoundChannel och lägger till stöd för paus och fortsatt uppspelning.
com/example/programmingas3/podcastplayer/URLService.as	En ActionScript-klass som hämtar data om en extern URL.
playerconfig.xml	En XML-fil som innehåller en lista över RSS-matningar som representerar poddsändningskanaler.
comp/example/programmingas3/utils/DateUtil.as	Klass som används för enkel datumformatering (endast Flash).

Läsa RSS-data för en poddsändningskanal

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet Podcast Player startar med att läsa information om ett antal poddsändningskanaler och deras avsnitt:

1. Först läser programmet en XML-konfigureringsfil som innehåller en lista över poddsändningskanaler och visar listan över kanalerna för användaren.

2. När användaren väljer en av poddsändningskanalerna, läses RSS-matningen för kanalen och en lista över kanalavsnitten visas.

I det här exemplet används en URLLoader-hjälpklass för att hämta textbaserad data från en extern plats eller en lokal fil. Podcast Player skapar först ett URLLoader-objekt för att få en lista över RSS-matningar i XML-format från filen playerconfig.xml. Sedan, när användaren väljer en specifik matning i listan, skapas ett nytt URLLoader-objekt för att läsa RSS-data från URL:n för den matningen.

Förenklad ljudinläsning och uppspelning med klassen SoundFacade

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ljudarkitekturen i ActionScript 3.0 är kraftfull men komplex. Program som endast behöver grundläggande funktioner för ljudinläsning och uppspelning kan använda en klass som döljer en del av komplexiteten genom att tillhandahålla en enklare uppsättning av metodanrop och händelser. I programvaruvärlden för designmönster kallas en sådan klass för *facade*.

Klassen SoundFacade presenterar ett enda gränssnitt för att utföra följande uppgifter:

- Läsa in ljudfiler med ett Sound-objekt, ett SoundLoaderContext-objekt och klassen SoundMixer
- Spela upp ljudfiler med Sound-objektet och SoundChannel-objektet
- Skicka händelser för uppspelningsförloppet
- Pausa och återuppta uppspelning av ljudet med Sound-objektet och SoundChannel-objektet

Klassen SoundFacade försöker erbjuda de flesta funktionerna som finns i ActionScript-ljudklasserna men inte lika komplicerade.

I följande kod visas klassdeklarationen, klassegenskaperna och konstruktormetoden `SoundFacade()`:

```
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;

        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

Klassen SoundFacade utökar klassen EventDispatcher så att den kan skicka sina egna händelser. Klasskoden deklarerar först egenskaper för ett Sound-objekt och ett SoundChannel-objekt. Klassen lagrar också URL-värdet för ljudfilen och en `bufferTime`-egenskap som används vid direktuppspelning av ljudet. Dessutom accepteras några booleska parametervärden som påverkar inläsnings- och uppspelningsbeteendena:

- Parametern `autoLoad` anger för objektet att ljudinläsningen ska starta så fort som det här objektet har skapats.
- Parametern `autoPlay` indikerar att ljuduppspelningen ska starta så fort som tillräckligt med ljuddata har lästs in. Om det är ett direktuppspelat ljud, börjar uppspelningen så fort som tillräcklig mängd data enligt specifikationen i `bufferTime`-egenskapen har lästs in.

- Med `streaming`-parametern indikeras att ljudfilen kan börja spelas upp innan inläsningen har slutförts.

Parametern `bufferTime` har standardvärdet -1. Om konstruktormetoden identifierar ett negativt värde i `bufferTime`-parametern, får `bufferTime`-egenskapen värdet från `SoundMixer.bufferTime`. Det här låter programmet valfritt ange standardvärde för det globala `SoundMixer.bufferTime`-värdet.

Om `autoLoad`-parametern har värdet `true`, anropar konstruktormetoden omedelbart följande `load()`-metod för att starta inläsningen av ljudfilen:

```
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, context);
}
```

`load()`-metoden skapar ett nytt Sound-objekt och lägger sedan till avlyssnare för alla viktiga ljudhändelser. Sedan aktiveras Sound-objektet att läsa in ljudfilen och använda SoundLoaderContext-objektet att föra över `bufferTime`-värdet.

Eftersom `url`-egenskapen kan ändras, kan en SoundFacade-instans användas för att spela upp olika ljudfiler i följd: Ändra `url`-egenskapen och anropa `load()`-metoden så läses den nya ljudfilen in.

Följande tre händelseavlyssningsmetoder visar hur SoundFacade-objekt följer inläsningsförloppet och avgör när uppspelning av ljudet ska starta:

```
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();
    }
}
```

Metoden `onLoadOpen()` körs när ljudinläsningen startar. Om ljudet kan spelas upp i direktuppspelningsläge, sätter `onLoadComplete()`-metoden `isReadyToPlay`-flaggan till `true` direkt. Med `isReadyToPlay`-flaggan fastställs om programmet kan starta ljuduppspelningen kanske som svar på en användaråtgärd som att klicka på Spela upp. Klassen SoundChannel hanterar buffringen av ljuddata, så det är inte nödvändigt att särskilt kontrollera om tillräckligt med data har lästs in innan `play()`-metoden anropas.

Metoden `onLoadProgress()` körs periodvis under inläsningsförloppet. Det skickar helt enkelt en klon av sitt ProgressEvent-objekt att användas av koden som använder det här SoundFacade-objektet.

När ljuddata har lästs in fullständigt, körs `onLoadComplete()`-metoden och anropar `play()`-metoden efter ljud som inte är direktuppspelade om det behövs. Själva `play()`-metoden visas nedan.

```
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();
        }
    }
}
```

`play()`-metoden anropar `Sound.play()`-metoden om ljudet är klart att spelas upp. Det slutliga SoundChannel-objektet lagras i `sc`-egenskapen. `play()`-metoden skapar sedan ett Timer-objekt som används för att skicka förloppshändelser för uppspelningen med regelbundet intervall.

Visa ljuduppspelningsförlopp

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att skapa ett Timer-objekt för att köra uppspelningsövervakning är en invecklad åtgärd som du endast ska behöva skriva kod för en gång. Med inkapslad Timer-logik i en återanvändbar klass som SoundFacade kan programmet avlyssna samma typ av förloppshändelser när ett ljud läses in och när det spelas upp.

Timer-objektet som skapades av `SoundFacade.play()`-metoden skickar en TimerEvent-instans varje sekund. Metoden `onPlayTimer()` nedan körs varje gång en ny TimerEvent anländer:

```
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()`-metoden implementerar tekniken för beräkning av storleken som beskrivs i avsnittet ”Övervaka uppspelning” på sidan 452. Sedan skapas en ny ProgressEvent-instans med händelsetypen `SoundFacade.PLAY_PROGRESS`, med `bytesLoaded`-egenskapen satt till aktuell position för SoundChannel-objektet och `bytesTotal`-egenskapen satt till den beräknade längden för ljuddata.

Pausa och återuppta uppspelning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

`SoundFacade.play()`-metoden som visades tidigare accepterar en `pos`-parameter som motsvarar en startposition i ljuddata. Om `pos`-värdet är noll startar uppspelningen av ljudet från början.

`SoundFacade.stop()`-metoden accepterar också en `pos`-parameter enligt nedan:

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

När `SoundFacade.stop()`-metoden anropas, anger den `pausePosition`-egenskapen så att programmet vet var spelhuvudet ska placeras om användaren vill återuppta uppspelningen av samma ljud.

Metoderna `SoundFacade.pause()` och `SoundFacade.resume()` som visas nedan startar `SoundFacade.stop()`- respektive `SoundFacade.play()`-metoderna och överför ett `pos`-parametervärde varje gång.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

`pause()`-metoden överför aktuellt `SoundChannel.position`-värde till `play()`-metoden, vilken lagrar värdet i egenskapen `pausePosition`. Med `resume()`-metoden startas uppspelningen av samma ljud igen med värdet i `pausePosition` som referenspunkt.

Utökning av Podcast Player-exemplet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I det här exemplet presenteras Podcast Player-grunder som visar hur den återanvändningsbara klassen SoundFacade används. Du kan lägga till andra funktioner för att öka användbarheten av det här programmet, inklusive följande:

- Lagra listan över matningar och användarinformation om varje avsnitt i en SharedObject-instans, som kan användas nästa gång användaren kör programmet.
- Låta användaren lägga till sin egen RSS-matning i listan över poddsändningskanaler.
- Komma ihåg placeringen av spelhuvudet när användaren stoppar eller lämnar ett avsnitt så att det kan återstartas från den positionen nästa gång användaren kör programmet.
- Ladda ned MP3-filer med avsnitt för lyssning offline när användaren inte är ansluten till Internet.
- Lägga till prenumerationsfunktioner som periodvis söker efter nya avsnitt i en poddsändningskanal och uppdaterar avsnittslistan automatiskt.
- Lägga till poddsändningssökning och visningsfunktioner med ett API från en poddsändningstjänst som Odeo.com.

Kapitel 25: Arbeta med video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash-video är en av standardteknikerna på Internet. Men det traditionella sättet att presentera video med en rektangulär skärm och en förloppsindikator med kontrollknappar är endast ett av de möjliga sätten att använda video. Med ActionScript har du finjusterad åtkomst och kontroll över videoinläsning, presentation och uppspelning.

Grunderna för video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En viktig funktion i Adobe® Flash® Player och Adobe® AIR™ är möjligheten att visa och ändra videoinformation med ActionScript på samma sätt som du kan ändra annat visuellt innehåll som bilder, animeringar, text och så vidare. När du skapar en Flash Video-fil (FLV) i Adobe Flash CS4 Professional har du möjligheten att välja ett skal som omfattar vanliga uppspelningskontroller. Men du behöver inte begränsa dig till de tillgängliga alternativen. Med ActionScript har du detaljerad kontroll över inläsning, visning och uppspelning av video, vilket betyder att du kan skapa ditt eget videouppspelningsskal eller använda videon på något mindre traditionellt sätt. När du arbetar med video i ActionScript arbetar du med en kombination av flera klasser:

- Klassen Video: Den klassiska rutan för videoinnehåll på scenen är en instans av klassen Video. Klassen Video är ett visningsobjekt, vilket innebär att det kan ändras på samma sätt som andra visningsobjekt, t.ex. när det gäller placering, färgomformning, filter och blandningslägen.
- Klassen StageVideo: Klassen Video använder vanligtvis programvaruavkodning och -återgivning. Om GPU-maskinvaruacceleration är tillgänglig på en enhet kan du utnyttja maskinvaruaccelererad presentation genom att växla till klassen StageVideo. StageVideo-API:t innehåller en uppsättning händelser som talar om för koden när det är lämpligt att växla mellan StageVideo- och Video-objekt. Scenvideo innebär vissa mindre begränsningar på videouppspelning. Om begränsningarna accepteras i programmet kan du implementera StageVideo API:et. Se ”Riktlinjer och begränsningar” på sidan 514.
- Klassen NetStream: När du läser in en videofil som ska styras av ActionScript, används en NetStream-instans för att representera källan för videoinnehållet, i det här fallet ett videodataflöde. När du använder en NetStream-instans måste du också använda ett NetConnection-objekt, som är anslutningen till videofilen (som den tunnel som videodata matas igenom).
- Klassen Camera: När du arbetar med videodata från en kamera som är ansluten till användarens dator, representerar Camera-instansen källan för videoinnehållet, användarens kamera och de videodata som blir tillgängliga med den. En nyhet i Flash Player 11.4 och AIR 3.4, är att du kan använda en kamera för att skicka bilder till StageVideo.

När du läser in extern video kan du läsa in filen från en standardwebbserver för progressiv nedladdning eller arbeta med direktuppspelad video som levereras av en specialserver som Adobes Flash® Media Server.

Viktiga termer och begrepp

Referenspunkt En markör som kan placeras vid ett specifikt tidsmoment i en videofil, t.ex. för att fungera som ett bokmärke för att hitta tidpunkten eller för att lägga in ytterligare data som hör till tidsmomentet.

Kodning Processen för att hämta videodata i ett format och konvertera det till ett annat videodataformat; t.ex. hämta en källvideo med hög upplösning och konvertera den till ett format som passar för leverans över Internet.

Bildruta Ett enstaka segment av videoinformation. Varje bildruta är som en stillbild och representerar en ögonblicksbild av ett tidsmoment. Genom att spela upp bildrutor i följd med hög hastighet erhålls illusionen av rörelse.

Nyckelbildruta En videobildruta som innehåller den fullständiga informationen för bildrutan. De andra bildrutorna som följer efter en nyckelbildruta innehåller endast information om hur de skiljer sig från nyckelbildrutan, i stället för att innehålla fullständig information om bildrutan.

Metadata Information om en videofil som bäddats in i videofilen och hämtas när videon har lästs in.

Progressiv hämtning När en videofil levereras från en standardwebbserver, läses videodata in med progressiv hämtning, vilket innebär att videoinformationen läses in sekventiellt. Fördelen med det är att videon kan börja spelas upp innan hela filen har hämtats, men å andra sidan kan du inte hoppa fram till en del av videon som inte har lästs in.

Direktuppspelning En speciell videoserver kan användas som ett alternativ till progressiv nedladdning för att leverera video över Internet med hjälp av teknik som kallas direktuppspelning (ibland kallad ”true streaming"). Vid direktuppspelning laddas aldrig hela videon ned på en gång i användarens dator. För att öka nedladdningshastigheten behöver datorn endast en del av den totala videoinformationen vid tillfället. Eftersom en specialserver styr leveransen av videoinnehållet, är alla delar av videon åtkomliga hela tiden i stället för att du behöver vänta på att den ska laddas ned innan du kan nå den.

Videoformat

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Förutom Adobe FLV-videoformatet har Flash Player och Adobe AIR stöd för ljud och video som kodas i H.264 i HE-AAC inifrån de vanliga MPEG-4-filformaten. Dessa format direktuppspelar video med hög kvalitet till lägre bithastighet. Utvecklare kan skapa och leverera avancerat videoinnehåll med standardverktyg som Adobe Premiere Pro och Adobe After Effects.

Typ	Format	Behållare
Video	H.264	MPEG-4: MP4, M4V, F4V, 3GPP
Video	Sorenson Spark	FLV-fil
Video	ON2 VP6	FLV-fil
Ljud	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4:MP4, M4V, F4V, 3GPP
Ljud	Mp3	Mp3
Ljud	Nellymoser	FLV-fil
Ljud	Speex	FLV-fil

Fler hjälpavsnitt

Flash Media Server: Supported codecs

Adobe HTTP Dynamic Streaming

Koda video för mobilenheter

AIR på Android kan avkoda ett stort antal H.264-videofilmer. Det är emellertid bara ett fåtal H.264-videofilmer som får en jämn uppspelning på mobiltelefoner. Detta beror på att många mobiltelefoner har begränsad processorkapacitet. Adobe Flash Player för mobilenheter kan avkoda H.264-videofilmer med hjälp av inbyggd maskinvaruacceleration. Den här avkodningen ger bättre kvalitet och mindre strömförbrukning.

H.264-standarden har stöd för flera kodningstekniker. Det är bara kraftfulla, högklassiga enheter som klarar att spela upp video med komplexa profiler och nivåer jämnt. De flesta enheter kan dock spela upp video som kodats med baslinjeprofilen. På mobilenheter är maskinvaruacceleration tillgängligt för en underuppsättning av de här teknikerna. Profilen och nivåparametrarna avgör vilken underuppsättning med tekniker och inställningar som används av kodaren. För utvecklare innebär det att videon kodas i en bestämd upplösning, som spelas upp bra på de flesta enheter.

Vilka upplösningar som drar nytta av maskinvaruacceleration varierar från enhet till enhet, men de flesta enheter har stöd för följande standardupplösningar.

Proportioner	**Rekommenderade upplösningar**		
4:3	640 × 480	512 × 384	480 × 360
16:9	640 × 360	512 x 288	480 × 272

Obs! *Flash Player har stöd för alla nivåer och profiler i H.264-standarden. Om du följer de här råden används maskinvaruacceleration på de flesta enheter, och användaren får en bättre upplevelse. Du måste naturligtvis inte följa råden.*

En detaljerad diskussion och kodningsinställningar i Adobe Media Encoder CS5 finns i Rekommendationer för kodning av H.264-video för Flash Player 10.1 på mobilenheter.

Obs! *I iOS kan bara video som kodats med Sorenson Spark- och On2 VP6-kodekar spelas upp med klassen Video. Du kan spela upp H.264-kodad video på enhetens videospelare genom att öppna webbadressen till videofilmen med funktionen* `flash.net.navigateToURL()`*. Du kan också spela upp H.264-video med taggen <video> på en HTML-sida som visas i ett StageWebView-objekt.*

Flash Player- och AIR-kompatibilitet med kodade videofiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player 7 har stöd för FLV-filer som har kodats med videokodek-programmet Sorenson™ Spark™. Flash Player 8 har stöd för FLV-filer som har kodats med Sorenson Spark- eller On2 VP6-kodaren i Flash Professional 8. Videokodek-programmet On2 VP6 har stöd för en alfakanal.

Flash Player 9.0.115.0 och senare versioner stöder filer som härleds från det vanliga MPEG-4-behållarformatet. Dessa filer omfattar F4V, MP4, M4A, MOV, MP4V, 3GP och 3G2, och de innehåller H.264-video eller HE-AAC v2-kodat ljud, eller både och. H.264 ger högre videokvalitet vid lägre bithastigheter jämfört med samma kodningsprofil i Sorenson eller On2. HE-AAC v2 är ett tillägg till AAC, ett standardljudformat som definieras i videostandarden MPEG-4. HE-AAC v2 använder tekniker för SBR (Spectral Band Replication) och PS (Parametric Stereo) för att öka kodningseffektiviteten vid låga bithastigheter.

De koder som stöds visas i följande tabell: Här visas också motsvarande SWF-filformat och vilka versioner av Flash Player och AIR som behövs för att spela upp dem:

Kodek	Version av SWF-filformat (tidigaste publicerad version)	Flash Player och AIR (tidigaste version som krävs för uppspelning)
Sorenson Spark	6	Flash Player 6, Flash Lite 3
On2 VP6	6	Flash Player 8, Flash Lite 3. Det är bara Flash Player 8 och senare versioner som har stöd för publicering och uppspelning av On2 VP6-video.
H.264 (MPEG-4 Part 10)	9	Flash Player 9 uppdatering 3, AIR 1.0
ADPCM	6	Flash Player 6, Flash Lite 3
Mp3	6	Flash Player 6, Flash Lite 3
AAC (MPEG-4 Part 3)	9	Flash Player 9 uppdatering 3, AIR 1.0
Speex (ljud)	10	Flash Player 10, AIR 1.5
Nellymoser	6	Flash Player 6

Adobe-videofilformaten F4V och FLV

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Adobe innehåller videofilformaten F4V och FLV för direktuppspelning av innehåll i Flash Player och AIR. En fullständig beskrivning av dessa videofilformat finns på www.adobe.com/go/video_file_format_se.

Videofilformatet F4V

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Från och med Flash Player uppdatering 3 (9.0.115.0) och AIR 1.0 har Flash Player och AIR stöd för Adobe F4V-videoformatet, som baseras på ISO MP4-formatet. Delmängder av formatet har stöd för olika funktionen. Flash Player måste ha en giltig F4V-fil för att kunna börja med någon av följande rutor på den översta nivån:

- ftyp

  Rutan ftyp identifierar funktionerna som programmet måste ha stöd för, för att kunna spela upp ett visst filformat.

- moov

  Rutan moov är huvudet i en F4V-fil. Den innehåller en eller flera andra rutor som i sin tur innehåller andra rutor som definierar strukturen i F4V-data. En F4V-fil får bara innehålla en moov-ruta.

- mdat

  En mdat-ruta innehåller datanyttolasten för F4V-filen. En FV-fil innehåller bara en mdat-ruta. En moov-ruta måste också finnas i filen eftersom mdat-rutan inte har någon betydelse på egen hand.

F4V-filer har stöd för flerbyteheltal i byteordningen big-endian, där den mest signifikanta byten finns först, vid den lägsta adressen.

Videofilformatet FLV

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Filformatet Adobe FLV innehåller kodade ljud- och videodata för leverans med Flash Player. Du kan använda en kodare, till exempel Adobe Media Encoder eller Sorenson™ Squeeze, om du vill konvertera en QuickTime- eller Windows Media-videofil till en FLV-fil.

Obs! *Du kan skapa FLV-filer genom att importera video till Flash och exportera den som en FLV-fil. Du kan använda plugin-programmet FLV Export för att exportera FLV-filer från videoredigeringsprogram som stöds. Om du vill läsa in FLV-filer från en webbserver registrerar du filnamnstillägget och MIME-typen hos webbservern. Läs mer i dokumentationen till webbservern. MIME-typen för FLV-filer är* `video/x-flv`*. Mer information finns i ”Konfigurera FLV-filer för värdskap på en server” på sidan 506.*

Mer information om FLV-filer finns i ”Avancerad information för videofiler” på sidan 505.

Extern kontra inbäddad video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Externa videofiler kräver viss kapacitet som inte är tillgänglig när du använder importerad video:

- Långa videoklipp kan användas i programmet utan att uppspelningen går långsammare. Externa videofiler använder cachelagrat minne, vilket innebär att stora filer lagras i mindre delar och du har åtkomst till dem dynamiskt. Därför behöver externa F4V- och FLV-filer mindre minne än inbäddade videofiler.
- En extern videofil kan ha en annan bildrutehastighet än SWF-filen som filen spelas upp i. Du kan t.ex. ange bildrutehastigheten 30 bildrutor per sekund (fps) för SWF-filen och 21 fps för videon. Den här inställningen ger dig bättre kontroll över videon än inbäddad video, och den ger en jämn videouppspelning. Du kan också spela upp videofiler med olika bildrutehastigheter utan att behöva ändra befintligt SWF-filinnehåll.
- Med externa videofiler avbryts inte uppspelningen av SWF-innehåll medan videofilen läses in. Med importerade videofiler avbryts ibland dokumentuppspelningen för att vissa funktioner ska utföras, t.ex. gå till en CD-skiva. Videofiler kan utföra funktioner oberoende av SWF-innehållet, utan att uppspelningen avbryts.
- Det är enklare att hämta videoinnehåll med externa FLV-filer, eftersom du kan använda händelsehanterare för att hämta metadata för videon.

Klassen Video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen Video kan du visa direktuppspelad video i ett program utan att bädda in videon i SWF-filen. Du kan spela in och spela upp direktvideo med metoden `Camera.getCamera()`. Du kan även använda klassen Video för att spela upp videofiler via HTTP eller från det lokala filsystemet. Det finns flera olika sätt att använda video i dina projekt:

- Läsa in en videofil dynamiskt med klasserna NetConnection och NetStream och visa videon i ett Video-objekt.
- Spela in data från användarens kamera. Mer information finns i ”Arbeta med kameror” på sidan 521.
- Använda FLVPlayback-komponenten.
- Använda kontrollen VideoDisplay.

Obs! *Instanser av ett videoobjekt på scenen är instanser av klassen Video.*

Även om klassen Video finns i paketet flash.media ärver den från klassen flash.display.DisplayObject. Därför gäller alla visningsobjektfunktioner, till exempel matrisomformningar och filter, även för Video-instanser.

Mer information finns i ”Ändra visningsobjekt” på sidan 173, ”Arbeta med geometri” på sidan 210 och ”Filtrera visningsobjekt” på sidan 268.

Läsa in videofiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att läsa in video med klasserna NetStream och NetConnection är en flerstegsprocess. Stegen med att lägga till Video-objektet i visningslistan, koppla NetStream-objektet till Video-instansen och anropa NetStream-objektets `play()`-metod bör utföras i den angivna ordningen:

1. Skapa ett NetConnection-objekt. Om du ansluter till en lokal videofil eller någon som inte använder en server som Adobes Flash Media Server 2 skickar du `null` till metoden `connect()` om du vill spela upp videofiler från en HTTP-adress eller lokal enhet. Om du ansluter till en server anger du parametern som URI-adressen för programmet som innehåller videofilen på servern.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. Skapa ett nytt Video-objekt som visar videon och lägg till det i visningslistan för scenen, enligt följande kodfragment:

```
var vid:Video = new Video();
addChild(vid);
```

3. Skapa ett NetStream-objekt och skicka NetConnection-objektet som ett argument till konstruktorn. Följande kodfragment kopplar ett NetStream-objekt till NetConnection-instansen och ställer in händelsehanterare för strömmen:

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);

function netStatusHandler(event:NetStatusEvent):void
{
    // handle netStatus events, described later
}

function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

4. Koppla NetStream-objektet till Video-objektet med Video-objektets `attachNetStream()`-metod, enligt följande kodfragment:

```
vid.attachNetStream(ns);
```

5. Anropa NetStream-objektets `play()`-metod med videofilens webbadress (url) som ett argument för att starta uppspelning av videon. Följande kodfragment läser in och spelar upp en videofil med namnet video.mp4 i samma katalog som SWF-filen:

```
ns.play("video.mp4");
```

Fler hjälpavsnitt

Flex: Spark VideoPlayer control

spark.components.VideoDisplay

Styra videouppspelning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I klassen NetStream finns fyra huvudmetoder för styrning av videouppspelning:

`pause()`: Pausar uppspelningen av ett videoflöde. Om videon redan är pausad när du anropar metoden händer ingenting.

`resume()`: Återupptar uppspelningen av ett pausat videoflöde. Om videoklippet redan spelas upp när du anropar metoden händer ingenting.

`seek()`: Söker efter nyckelbildrutan som är närmast den angivna platsen (en förskjutning i sekunder från direktuppspelningens början).

`togglePause()`: Pausar eller återupptar uppspelningen av ett videoflöde.

Obs! *Det finns ingen `stop()`-metod. För att stoppa direktuppspelningen måste du pausa uppspelningen och söka efter början av videoflödet.*

Obs! *Det går inte att återuppta uppspelningen med `play()`-metoden, den används för inläsning av videofiler.*

I följande exempel visas hur du styr en video med flera olika knappar. För att köra följande exempel skapar du ett nytt dokument och lägger till fyra knappinstanser på din arbetsyta (`pauseBtn`, `playBtn`, `stopBtn` och `togglePauseBtn`):

```
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
{
    // ignore error
}

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
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

Om du klickar på `pauseBtn`-knappen när videon spelas upp pausas videofilen. Om videon redan har pausats händer ingenting när du klickar på knappen. Om du klickar på `playBtn`-knappen återtas videouppspelningen om den var pausad tidigare, annars händer ingenting om videon redan håller på att spelas upp.

Identifiera slutet av ett videoflöde

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att kunna avlyssna början och slutet i ett videoflöde, måste du lägga till en händelseavlyssnare i NetStream-instansen som ska avlyssna `netStatus`-händelsen. I följande kod visas hur de olika koderna avlyssnas i videouppspelningen:

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

Den föregåenden koden genererar följande utdata:

```
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
```

De två koder som du speciellt vill avlyssna är ”NetStream.Play.Start” och ”NetStream.Play.Stop", vilka signalerar början och slutet av en videouppspelning. I följande kodutdrag används en switch-programsats för att filtrera de här två koderna och spåra ett meddelande:

```
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;
    }
}
```

Genom att avlyssna `netStatus`-händelsen (`NetStatusEvent.NET_STATUS`), kan du skapa en videospelare som läser in nästa video i en spelningslista när uppspelningen av den aktuella videon har slutat.

Spela upp video i helskärmsläge

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med Flash Player och AIR kan du skapa ett helskärmsprogram för videouppspelning, med stöd för videoskalning till helskärm.

För AIR-innehåll som körs i helskärmsläge på skrivbordet är systemets skärmsläckare och energisparalternativ inaktiverade under uppspelningen tills videoindata upphör eller användaren avbryter helskärmsläget.

Mer information om hur du använder helskärmsläget finns i avsnittet ”Arbeta i helskärmsläge” på sidan 167.

Aktivera helskärmsläget för Flash Player i en webbläsare

Innan du kan implementera helskärmsläge för Flash Player i en webbläsare måste du aktivera det via programmets publiceringsmall. Mallarna som använder helskärmsläge omfattar `<object>`- och `<embed>`-taggar som innehåller en `allowFullScreen`-parameter. I följande exempel visas `allowFullScreen`-parametern i en `<embed>`-tagg.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

Välj Arkiv -> Publiceringsinställningar i Flash och välj sedan mallen Endast Flash - Tillåt helskärm på fliken HTML i dialogrutan Publiceringsinställningar.

I Flex ser du till att HTML-mallen omfattar `<object>`- och `<embed>`-taggarna som stöder helskärm.

Initiera helskärmsläge

När det gäller Flash Player-innehåll som körs i en webbläsare aktiverar du helskärmsläget för video som svar på en musklickning eller tangenttryckning. Du kan till exempel initiera helskärmsläget när användaren klickar på en knapp med etiketten Helskärm eller väljer kommandot Helskärm på snabbmenyn. Om du vill svara användaren lägger du till en händelseavlyssnare i objektet som åtgärden utförs på. Följande kod lägger till en händelseavlyssnare för en knapp som användaren klickar på för att aktivera helskärmsläget:

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

Koden initierar helskärmsläge genom att `Stage.displayState`-egenskapen tilldelas värdet `StageDisplayState.FULL_SCREEN`. Den här koden skalar hela scenen till helskärm med videoskalningen i förhållande till utrymmet som upptas på scenen.

Med egenskapen `fullScreenSourceRect` kan du ange att ett visst område på scenen ska skalas till helskärm. Först definierar du rektangeln som du vill skala till helskärm. Tilldela den sedan till egenskapen `Stage.fullScreenSourceRect`. Den här versionen av funktionen `fullScreenButtonHandler()` lägger till två extra kodrader som skalar enbart videon till helskärm.

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

I det här exemplet anropas en händelsehanterare som svar på en musklickning, men tekniken för att övergå till helskärmsläge är samma för både Flash Player och AIR. Definiera rektangeln som du vill skapa och ställ sedan in egenskapen `Stage.displayState`. Mer information finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Det fullständiga exemplet, som följer, lägger till kod som skapar anslutningen och NetStream-objektet för videon och börjar spela upp den.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
```

```
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width,
video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

Funktionen `onMetaData()` är en callback-funktion som hanterar videometadata, om sådana finns. En callback-funktion är en funktion som körtidsbiblioteket anropar som svar på någon typ av förekomst eller händelse. I det här exemplet är funktionen `onMetaData()` en stub som innehåller det som krävs för att ange funktionen. Mer information finns i ”Skriva callback-metoder för metadata och referenspunkter” på sidan 487

Lämna helskärmsläge

En användare kan lämna helskärmsläget med något av kortkommandona, till exempel Esc-tangenten. Du kan avbryta helskärmsläget i ActionScript genom att ge `Stage.diplayState`-egenskapen värdet `StageDisplayState.NORMAL`. Koden i följande exempel avbryter helskärmsläget när händelsen NetStream.Play.Stop `netStatus` inträffar.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

Maskinvaruacceleration i helskärmsläge

När du skalar om ett rektangulärt område av scenen till helskärmsläge används maskinvaruacceleration i Flash Player eller AIR om det är tillgängligt och aktiverat. Körningsmiljön använder datorns bildskärmskort så att det ska gå snabbare att skala videon, eller en del av scenen, till helskärmsstorlek. Under sådana förhållanden kan det vara en fördel att för FlashPlayer-program använda klassen StageVideo istället för klassen Video (eller Camera, i Flash Player 11.4/AIR 3.4 och senare).

Mer information om maskinvaruacceleration i helskärmsläge finns i avsnittet ”Arbeta i helskärmsläge” på sidan 167. Mer information om StageVideo finns i ”Använda klassen StageVideo för maskinvaruaccelererad presentation” på sidan 512.

Direktuppspelning av videofiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill direktuppspela filer från Flash Media Server, kan du använda klasserna NetConnection och NetStream för anslutning till en extern serverinstans och spela upp ett specificerat flöde. Du specificerar en Real-Time Messaging Protocol RTMP-server genom att överföra önskad RTMP-URL, t.ex. ”rtmp://localhost/appName/appInstance” till `NetConnection.connect()`-metoden i stället för att föra över `null`. Om du vill spela upp en direktsänd eller inspelad ström från angiven Flash Media Server skickar du ett identifieringsnamn för direktsända data som publiceras av `NetStream.publish()` eller ett inspelat filnamn för uppspelning till `NetStream.play()`-metoden.

Skicka video till en server

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill skapa mer komplexa program med video- eller kameraobjekt, erbjuder Flash Media Server (FMS) en kombination av direktuppspelade mediemöjligheter och en utvecklingsmiljö för att skapa och leverera medieprogram till en bred publik. Med den här kombinationen kan utvecklare skapa program som Video on Demand, direktsända webbhändelser och MP3-direktuppspelning såväl som videobloggning, videomeddelanden och multimediechattmiljöer. Mer information finns i dokumentationen till Flash Media Server på www.adobe.com/go/learn_fms_docs_se.

Referenspunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan bädda in referenspunkter i en F4V- eller FLV-videofil under kodningen. Referenspunkter har tidigare bäddats in i filmer för att ge filmmaskinisten en visuell signal som anger att filmrullen börjar ta slut. I Adobe F4V- och FLV-videoformaten medför en referenspunkt att du kan utlösa en eller flera åtgärder i programmet när dessa förekommer i videoströmmar.

Du kan använda flera olika typer av referenspunkter i Flash Video. Med hjälp av ActionScript kan du samverka med referenspunkterna som du bäddar in i en videofil när du skapar den.

- Navigeringsreferenspunkter: Du bäddar in navigeringsreferenspunkter i videouppspelningen och i metadatapaketet när du kodar videofilen. Du använder navigeringsreferenspunkter för att användaren ska kunna söka efter en viss del av en fil.
- Händelsereferenspunkter: Du bäddar in händelsereferenspunkter i videouppspelningen och i metadatapaketet när du kodar videofilen. Du kan skriva kod som hanterar händelser som utlöses vid angivna punkter under videouppspelning.
- ActionScript-referenspunkter: ActionScript-referenspunkter finns bara för Flash-komponenten FLVPlayback. ActionScript-referenspunkter är externa referenspunkter som du skapar och får tillgång till med hjälp av ActionScript-kod. Du kan skriva kod som utlöser dessa referenspunkter i relation till videouppspelningen. Dessa referenspunkter är mindre exakta än inbäddade referenspunkter (upp till en tiondels sekund), eftersom videospelaren spårar dem separat. Om du planerar att skapa ett program där du vill att användarna ska gå till en referenspunkt, bör du skapa och bädda in referenspunkter när du kodar filen i stället för att använda ActionScript-referenspunkter. Du bör bädda in referenspunkterna i FLV-filen eftersom de är mera exakta.

Navigeringsreferenspunkter skapar en nyckelbildruta på den angivna referenspunktspositionen, så att du kan använda kod för att flytta videospelarens spelhuvud till den här positionen. Du anger vissa punkter i en videofil där du vill att användarna ska söka. Din video kan t.ex. ha flera kapitel eller segment och du kan styra videon genom att bädda in navigeringsreferenspunkter i videofilen.

Mer information om kodning av Adobe-videofiler med referenspunkter finns i ”Bädda in referenspunkter” i *Använda Flash*.

Du kan hämta referenspunktsparametrar genom att skriva ActionScript. Referenspunktsparametrar är en del av händelseobjektet som tas emot via callback-hanteraren

Du kan använda `NetStream.onCuePoint`-händelsehanteraren för att aktivera vissa åtgärder i koden när FLV-filen når en angiven referenspunkt.

Om du vill synkronisera en åtgärd för en referenspunkt i en F4V-videofil måste du hämta referenspunktsdata från callback-funktionen `onMetaData()` eller `onXMPData()` och utlösa referenspunkten med hjälp av klassen Timer i ActionScript 3.0. Mer information om F4V-referenspunkter finns i ”Använda onXMPData()” på sidan 498.

Mer information om hur du hanterar referenspunkter och metadata finns i ”Skriva callback-metoder för metadata och referenspunkter” på sidan 487.

Skriva callback-metoder för metadata och referenspunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan aktivera åtgärder i programmet när vissa referenspunkter nås eller vissa metadata mottas av spelaren eller när vissa referenspunkter påträffas. När dessa händelser inträffar måste du ange särskilda callback-metoder som händelsehanterare. Klassen NetStream anger följande metadatahändelser som kan inträffa under uppspelning: `onCuePoint` (enbart FLV-filer), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` och `onXMPData`.

Du måste skriva callback-metoder för dessa hanterare, annars kan fel uppstå i Flash-körningen. I följande kod spelas en FLV-fil upp med namnet video.flv i samma mapp som SWF-filen finns:

```
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);
```

Med föregående kod läses en lokal videofil in med namnet video.flv och den lyssnar efter den `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) som ska skickas. Den här händelsen skickas när ett undantag inträffar från den ursprungliga asynkrona koden. I det här fallet skickas den när en videofil innehåller metadata eller referenspunktsinformation och de rätta avlyssnarna inte har definierats. I den föregående koden hanteras `asyncError`-händelsen och felet ignoreras om du inte är intresserad av videofilens metadata eller referenspunktsinformationen. Om du hade en FLV-fil med metadata och flera referenspunkter visar `trace()`-funktionen följande felmeddelande:

```
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.
```

Felet inträffar för att NetStream-objektet inte kunde hitta någon callback-metod av typen `onMetaData` eller `onCuePoint`. Det finns flera sätt att definiera de här återanropsmetoderna i programmet.

Fler hjälpavsnitt

Flash Media Server: Handling metadata in streams

Ange NetStream-objektets client-egenskap som ett objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Genom att ange `client`-egenskapen som antingen ett objekt eller en underklass till NetStream, kan du omdirigera callback-metoderna `onMetaData` och `onCuePoint` eller helt ignorera dem. I följande exempel visas hur du kan använda ett tomt objekt för att ignorera callback-metoderna utan att avlyssna `asyncError`-händelsen:

```
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);
```

Om du vill avlyssna någon av callback-metoderna `onMetaData` eller `onCuePoint`t, måste du definiera metoder som hanterar dessa callback-metoder enligt exemplet i följande kodutdrag:

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

I föregående kod avlyssnas callback-metoden `onMetaData` och anropar `metaDataHandler()`-metoden som spårar en sträng. Om Flash-körningen påträffar en referenspunkt genereras inga fel även om någon callback-metod av typen `onCuePoint` inte har definierats.

Skapa en anpassad klass och definiera metoder för hantering av återanropsmetoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande kod anges NetStream-objektets `client`-egenskap som en anpassad klass, CustomClient, som definierar hanterare för callback-metoderna:

```
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);
```

Klassen CustomClient:

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

I klassen CustomClient definieras en hanterare för callback-hanteraren `onMetaData`. Om en referenspunkt påträffas och callback-hanteraren `onCuePoint` anropas, ska en `asyncError`-händelse (`AsyncErrorEvent.ASYNC_ERROR`) skickas som anger: ”flash.net.NetStream kunde inte starta callback-funktionen onCuePoint”. För att förhindra det här felet måste du antingen definiera callback-metoden `onCuePoint` i klassen CustomClient eller definiera en händelsehanterare för `asyncError`-händelsen.

Utöka klassen NetStream och lägg till metoder för hantering av callback-metoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande kod skapas en instans av klassen CustomNetStream som definieras i en senare kodlista:

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Följande kod definierar klassen CustomNetStream som utökar klassen NetStream, skapar det nödvändiga NetConnection-objektet och hanterar hanterarmetoderna för `onMetaData`- och `onCuePoint`-callback:

```
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");
        }
    }
}
```

Om du vill byta namn på `onMetaData()`- och `onCuePoint()`-metoderna i klassen CustomNetStream kan du använda följande kod:

```
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");
        }
    }
}
```

Utöka klassen NetStream och göra den dynamisk

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan utöka klassen NetStream och göra underklassen dynamisk så att callback-hanterarna `onCuePoint` och `onMetaData` kan läggas till dynamiskt. Det visas i följande kodexempel:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Klassen DynamicCustomNetStream:

```
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);
        }
    }
}
```

Även utan hanterare för callback-hanterarna `onMetaData` och `onCuePoint` inträffar inga fel eftersom klassen DynamicCustomNetStream är dynamisk. Om du vill definiera metoderna för callback-hanterarna `onMetaData` och `onCuePoint` kan du använda följande kod:

```
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");
}
```

Ange NetStream-objektets klientegenskap till detta

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du anger `client`-egenskapen till `this` söker programmet i aktuellt omfång efter `onMetaData()`- och `onCuePoint()`-metoder. Det visas i följande exempel:

```
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);
```

Om callback-hanterarna `onMetaData`- eller `onCuePoint` anropas och ingen metod finns som hanterar callback-funktionen, genereras inga fel. Om du vill hantera de här callback-hanterarna skapar du en `onMetaData()`- och `onCuePoint()`-metod i koden enligt följande kodutdrag:

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

Använda referenspunkter och metadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd NetStream-callback-metoderna om du vill fånga in och bearbeta referenspunkts- och metadatahändelser allt eftersom videon spelas upp.

Använda referenspunkter

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I tabellen nedan beskrivs callback-metoderna som du kan använda när du fångar in F4V- och FLV-referenspunkter i Flash Player och AIR.

Körning	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData

Körning	F4V	FLV
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

I följande exempel används en enkel `for..in`-slinga för att iterera igenom varje egenskap i `infoObject`-parametern som funktionen `onCuePoint()` tar emot. Den anropar funktionen trace() för att visa ett meddelande när den tar emot referenspunktsdata:

```
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]);
    }
}
```

Följande utdata visas:

```
parameters:
name: point1
time: 0.418
type: navigation
```

I den här koden används en av flera tekniker för att ange objektet som kör callback-metoden. Du kan använda fler tekniker. Mer information finns i ”Skriva callback-metoder för metadata och referenspunkter” på sidan 487.

Använda videometadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med funktionerna `OnMetaData()` och `OnXMPData()` får du åtkomst till metadatainformation i videofilen, bland annat referenspunkter.

Använda OnMetaData()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metadata innehåller information om videofilen, t.ex. varaktighet, bredd, höjd och bildrutehastighet. Vilken metadatainformation som läggs till i videofilen beror på vilket program du använder för att koda videofilen.

```
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]);
    }
}
```

Den föregående koden genererar följande utdata:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

Om videon saknar ljud returneras den ljudrelaterade metadatainformationen (som `audiodatarate`) `undefined` eftersom ingen ljudinformation har lagts till i metadata under kodningen.

I föregående kod visas inte referenspunktsinformationen. Om du vill visa metadata för referenspunkten kan du använda följande funktion som visar objekten rekursivt i ett objekt:

```
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 + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

Om du använder föregående kodutdrag för att spåra `infoObject`-parametern i `onMetaData()`-metoden skapas följande utdata:

```
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
```

I följande exempel visas metadata för en MP4-video. Det förutsätter att det finns ett TextArea-objekt med namnet `metaDataOut` som metadata ska skrivas till.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

Funktionen `onMetaData()` returnerar följande utdata för videon:

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

Använda informationsobjektet

I tabellen nedan visas möjliga värden för videometadata som skickas till callback-funktionen `onMetaData()` i det objekt som de tas emot i:

Parameter	Beskrivning
aacaot	AAC-ljudobjekttyp; 0, 1 eller 2 stöds.
avclevel	AVC IDC-nivånummer som 10, 11, 20, 21 och så vidare.
avcprofile	AVC-profilnummer som 55, 77, 100 och så vidare.
audiocodecid	En sträng som visar vilken ljudkodek som används (kodnings- och avkodningsteknik), till exempel .Mp3 eller mp4a.
audiodatarate	Ett tal som anger hastigheten som ljudet kodas med i kilobyte per sekund.
audiodelay	Ett tal som anger vilken tid i FLV-filen som är ”time 0” i den ursprungliga FLV-filen. Videoinnehållet måste fördröjas något för att synkronisera ljudet rätt.
canSeekToEnd	Ett booleskt värde som är `true` om FLV-filen är kodad med en nyckelbildruta på den sista bildrutan som möjliggör sökning ända till slutet av en progressivt nedladdad videofilm. Värdet är `false` om FLV-filen inte kodas med en nyckelbildruta på den sista bildrutan.
cuePoints	En array med objekt, en för varje referenspunkt som är inbäddad i FLV-filen. Värdet är odefinierat om FLV-filen inte innehåller några referenspunkter. Varje objekt innehåller följande egenskaper: • `type`: En sträng som anger typen av referenspunkt som antingen ”navigering” eller ”händelse”. • `name`: En sträng som är namnet på referenspunkten. • `time`: Ett tal som anger tiden i referenspunkten i sekunder med en noggrannhet på tre decimaler (millisekunder). • `parameters`: Ett valfritt objekt med namnvärdepar som bestäms av användaren när referenspunkterna skapas.
duration	Ett tal som anger varaktigheten för videofilen i sekunder.
framerate	Ett tal som anger bildrutehastigheten i FLV-filen.
height	Ett tal som anger FLV-filens höjd i pixlar.
seekpoints	En array som visar tillgängliga nyckelrutor som tidsstämplar i millisekunder. Valfritt.
tags	En array med nyckelvärdespar som representerar informationen i atomen ”ilst”, som är motsvarigheten till ID3-taggar för MP4-filer. iTunes använder dessa taggar. Kan användas för att visa grafik om sådan finns.
trackinfo	Objekt som omfattar information på alla spår i MP4-filen, även samplingens beskrivnings-ID.
videocodecid	En sträng som visar vilken kodekversion som videon kodas i. – till exempel för ”avc1” eller ”VP6F”

Parameter	Beskrivning
videodatarate	Ett tal som anger videodatahastigheten i FLV-filen.
videoframerate	MP4-videons bildhastighet.
width	Ett tal som anger FLV-filens bredd i pixlar.

I följande tabell visas giltiga värden för `videocodecid`-parametern:

videocodecid	Kodeknamn
2	Sorenson H.263
3	Screen-video (endast SWF version 7 och senare)
4	VP6 (endast SWF version 8 och senare)
5	VP6-video med alfakanal (endast SWF version 8 och senare)

I följande tabell visas giltiga värden för `audiocodecid`-parametern:

audiocodecid	Kodeknamn
0	ej komprimerad
1	ADPCM
2	Mp3
4	Nellymoser @ 16 kHz mono
5	Nellymoser 8 kHz mono
6	Nellymoser
10	AAC
11	Speex

Använda onXMPData()

Flash Player 10 och senare, Adobe AIR 1.5 och senare

Callback-funktionen `onXMPData()` får information om Adobe Extensible Metadata Platform (XMP) som bäddas in i Adobe F4V- eller FLV-videofilen. XMP-metadata omfattar referenspunkter samt andra videometadata. XMP-metadata stöds från och med Flash Player 10 och Adobe AIR 1.5 och i efterföljande versioner.

I följande exempel bearbetas referenspunktsdata i XMP-metadata:

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
```

```
                var strTrackName:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                var strFrameRateXML:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                nTracksFrameRate = Number(strFrameRate);

                strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

För en kort videofil med namnet startrekintro.f4v skapar det här exemplet följande trace-rader. Raderna visar referenspunktsdata för navigering och händelsereferenspunkter i XMP-metadata:

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

Obs! *I XMP lagras tid som DVA-ticks i stället för sekunder. Om du vill beräkna tiden för referenspunkten dividerar du starttiden med bildhastigheten. Exempel: Starttiden 7695905817600 dividerat med bildhastigheten 254016000000 är lika med 30:30.*

Om du vill visa fullständiga råa XMP-metadata, som omfattar bildhastighet, tar du bort kommentarsidentifierarna (//) som föregår den andra och tredje `trace()`-satsen i början av `onXMPData()`-funktionen.

Mer information om XMP finns på:

- http://partners.adobe.com/public/developer/xmp/topic.html
- http://www.adobe.com/devnet/xmp/

Använda bildmetadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelsen `onImageData` skickar bilddata i form av en byte-array genom en AMF0-datakanal. Data kan vara i JPEG-, PNG- eller GIF-format. Definiera en callback-metod av typen `onImageData()` om du vill bearbeta den här informationen, på samma sätt som du definierar callback-metoder för `onCuePoint` och `onMetaData`. Följande exempel får åtkomst till och visar bilddata med hjälp av callback-metoden `onImageData()`:

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

Använda textmetadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelsen `onTextData` skickar textdata genom en AMF0-datakanal. Alla textdata är i UTF-8-format och innehåller extra information om formatering baserat på textspecifikationen 3GP timed. Den här specifikationen definierar ett standardiserat textningsformat. Definiera en callback-metod av typen `onTextData()` om du vill bearbeta den här informationen, på samma sätt som du definierar callback-metoder för `onCuePoint` eller `onMetaData`. I följande exempel visar metoden `onTextData()` spårets ID-nummer och motsvarande spårtext.

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

Övervaka NetStream-aktivitet

Flash Player 10.3 och senare, Adobe AIR 2.7 och senare

Du kan övervaka en NetStream-aktivitet för att samla in den information som krävs för att understödja medieanvändningsanalys och rapporter. Den övervakningsfunktion som presenteras i detta avsnitt gör det möjligt för dig att skapa bibliotek för mediemätningar där det går att samla in data utan att stänga kopplingen till den speciella videospelaren där mediet visas. Detta gör att klientutvecklare kan välja speciella videospelare när de använder ditt bibliotek. Använd klassen NetMonitor för att övervaka aktiviteten och hur NetStream-objekt skapas i ett program. Klassen NetMonitor ger dig en lista med aktiva NetStreams vid en given tidpunkt. Dessutom skickas en händelse när ett NetStream-objekt skapas.

Beroende på vilken typ av media som spelas skickar ett NetStream-objekt händelserna som visas i tabellen nedan:

Händelse	Progressiv hämtning	RTMP-direktuppspelning	HTTP-direktuppspelning
NetStream.Play.Start	Ja	Ja	Nej
NetStream.Play.Stop	Ja	Ja	Nej
NetStream.Play.Complete	Ja	Ja	Nej
NetStream.SeekStart.Notify	Ja	Ja	Ja
NetStream.Seek.Notify	Ja	Ja	Ja
NetStream.Unpause.Notify	Ja	Ja	Ja
NetStream.Unpause.Notify	Ja	Ja	Ja
NetStream.Play.Transition	Används ej	Ja	Används ej
NetStream.Play.TransitionComplete	Används ej	Ja	Används ej
NetStream.Buffer.Full	Ja	Ja	Ja
NetStream.Buffer.Flush	Ja	Ja	Ja
NetStream.Buffer.Empty	Ja	Ja	Ja

NetStreamInfo-objektet som är associerat med en NetStream-instans lagrar även senaste metadata och XMP-dataobjekt som upptäckts i mediet.

När ett media spelas upp med HTTP-direktuppspelning, skickas inte NetStream.Play.Start, NetStream.Play.Stop och NetStream.Play.Complete eftersom programmet har fullständig kontroll över medieströmmen. En videospelare ska sammanfoga och skicka dessa händelser för HTTP-direktuppspelningar.

Detsamma gäller för NetStream.Play.Transition och NetStream.Play.TransitionComplete som varken inte skickas för progressiv nedladdning eller för HTTP-media. Dynamisk bithastighetsväxling är en RTMP-funktion. Om en videospelare där HTTP-direktuppspelning används har stöd för en liknande funktion, kan spelaren sammanfoga och skicka övergångshändelser.

Fler hjälpavsnitt

Adobe Developer Connection: Mäta videoförbrukning i Flash

Övervaka NetStream-händelser

Det finns två typer av händelser som tillhandahåller värdefulla användardata: `netStatus` och `mediaTypeData`. Dessutom kan en timer användas för att med jämna mellanrum logga positionen för NetStream-spelhuvudet.

`netStatus`-händelser ger information som du kan använda för att bestämma hur mycket av en direktuppspelning som en användare kunde se. Buffring och övergångshändelser för RTMFP-direktuppspelning resulterar även i en `netStatus`-händelse.

`mediaTypeData`-händelser tillhandahåller metadata och XMP-datainformation. Händelsen Netstream.Play.Complete skickas som en `mediaTypeData`-händelse. Även andra data inbäddade i direktuppspelningen är tillgängliga via `mediaTypeData`-händelser, inklusive referenspunkter, text och bilder.

I följande exempel visas hur du skapar en klass för att övervaka status- och datahändelser från aktiva NetStreams i ett program. Vanligtvis skulle en sådan klass skicka data som var intressant att analysera till en server för att samlas in.

```
package com.adobe.example
{
    import flash.events.NetDataEvent;
    import flash.events.NetMonitorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetMonitor;
    import flash.net.NetStream;

    public class NetStreamEventMonitor
    {
        private var netmon:NetMonitor;
        private var heartbeat:Timer = new Timer( 5000 );

        public function NetStreamEventMonitor()
        {
            //Create NetMonitor object
            netmon = new NetMonitor();
            netmon.addEventListener( NetMonitorEvent.NET_STREAM_CREATE, newNetStream );

            //Start the heartbeat timer
            heartbeat.addEventListener( TimerEvent.TIMER, onHeartbeat );
            heartbeat.start();
        }

        //On new NetStream
        private function newNetStream( event:NetMonitorEvent ):void
        {
            trace( "New Netstream object");
            var stream:NetStream = event.netStream;
            stream.addEventListener(NetDataEvent.MEDIA_TYPE_DATA, onStreamData);
            stream.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
        }

        //On data events from a NetStream object
        private function onStreamData( event:NetDataEvent ):void
        {

            var netStream:NetStream = event.target as NetStream;
            trace( "Data event from " + netStream.info.uri + " at " + event.timestamp );
            switch( event.info.handler )
            {
                case "onMetaData":
                    //handle metadata;
                    break;
                case "onXMPData":
                    //handle XMP;
                    break;
                case "onPlayStatus":
                    //handle NetStream.Play.Complete
                case "onImageData":
                    //handle image
                    break;
                case "onTextData":
                    //handle text
                    break;
                default:
                    //handle other events
```

```
            }
        }

        //On status events from a NetStream object
        private function onStatus( event:NetStatusEvent ):void
        {
            trace( "Status event from " + event.target.info.uri + " at " + event.target.time );
            //handle status events
        }
        //On heartbeat timer
        private function onHeartbeat( event:TimerEvent ):void
        {
            var streams:Vector.<NetStream> = netmon.listStreams();
            for( var i:int = 0; i < streams.length; i++ )
            {
                trace( "Heartbeat on " + streams[i].info.uri + " at " + streams[i].time );
                //handle heartbeat event
            }
        }

    }
}
```

Identifiera spelardomän

Den URL och domän som används för webbsidan där en användare tittar på medieinnehåll är inte alltid tillgänglig. Om det tillåts på värdwebbplatsen kan du använda klassen ExternalInterface för att få exakt rätt URL. Vissa webbplatser, som tillåter videospelare från tredje part, tillåter emellertid inte att ExternalInterface används. I dessa fall ska du hämta domänen för den aktuella sidan från egenskapen `pageDomain` i Security-klassen. Den fullständiga URL:en avslöjas inte eftersom man vill skydda användaren och hans/hennes sekretess.

Siddomänen är tillgäng via den statiska `pageDomain`-egenskapen i Security-klassen:

```
var domain:String = Security.pageDomain;
```

Avancerad information för videofiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande ämnen handlar om speciella frågor för arbete med FLV-filer.

Konfigurera FLV-filer för värdskap på en server

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du arbetar med FLV-filer måste du kanske konfigurera servern så att den kan arbeta med FLV-filformatet. MIME (Multipurpose Internet Mail Extensions) är en standardiserad dataspecifikation som du använder för att skicka andra typer än ASCII-filer över Internet-anslutningar. Webbläsare och e-postklienter har konfigurerats för att tolka olika MIME-typer så att de kan skicka och ta emot video, ljud, grafik och formaterad text. Om du vill läsa in FLV-filer från en webbserver måste du kanske registrera filnamnstillägget och MIME-typen hos webbservern, vilket beskrivs i webbserverdokumentationen. MIME-typen för FLV-filer är `video/x-flv`. Den fullständiga informationen för FLV-filtypen är:

- Mime-typ: video/x-flv
- Filnamnstillägg: .flv
- Obligatoriska parametrar: none
- Valfria parametrar: none
- Att tänka på vid kodning: FLV-filer är binära filer och vissa program kräver att program/oktett-stream-typer anges
- Säkerhetsfrågor: none
- Publicerad specifikation: www.adobe.com/go/video_file_format_se

Microsoft har ändrat det sätt på vilket direktuppspelningsmedier hanteras i webbservern IIS 6.0 (Microsoft Internet Information Services) jämfört med tidigare versioner. Tidigare versioner av IIS kräver inga ändringar för att direktuppspela Flash Video. I IIS 6.0, standardwebbservern som levereras med Windows 2003, kräver servern en MIME-typ för att identifiera att FLV-filer är direktuppspelningsmedia.

När SWF-filer som direktuppspelar externa FLV-filer placeras på en Microsoft Windows Server® 2003 och visas i en webbläsare, spelas filerna upp på rätt sätt, men FLV-videon direktuppspelas inte. Detta påverkar alla FLV-filer som har placerats på Windows 2003-servern, inklusive de filer du har skapat med tidigare versioner av utvecklingsverktyget i Flash, och Macromedia Flash Video Kit for Dreamweaver MX 2004 från Adobe. Dessa filer fungerar på rätt sätt om du testar dem på andra operativsystem.

Mer information om hur du konfigurerar Microsoft Windows 2003 och Microsoft IIS Server 6.0 för att direktuppspela FLV-video finns på www.adobe.com/go/tn_19439_se.

Använda lokala FLV-filer på Macintosh

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du försöker spela upp en lokal FLV-fil från en enhet som inte finns i systemet på en Apple® Macintosh®-dator genom att använda en sökväg med ett relativt snedstreck (/), spelas videon inte upp. Enheter som inte finns i systemet inkluderar, men begränsas inte till, CD-skivor, partitionerade hårddiskar, flyttningsbara lagringsmedier och anslutna lagringsenheter.

Obs! *Orsaken till detta är en begränsning i operativsystemet, inte en begränsning i Flash Player eller AIR.*

Om en FLV-fil ska kunna spelas upp från en enhet som inte finns i systemet på Macintosh, måste du ange en absolut sökväg med ett kolon (:) i stället för ett snedstreck (/). I följande lista visas skillnaden mellan dessa två sätt:

- Snedstrecksbaserad syntax**:** myDrive/myFolder/myFLV.flv
- Kolonbaserad syntax**:** (Mac OS®) myDrive:myFolder:myFLV.flv

Exempel på video: videojukebox

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande exempel skapas en enkel videojukebox som dynamiskt läser in en lista med videofilerna som spelas upp i ordningsföljd. Så här kan du skapa ett program som du kan använda för att bläddra igenom en serie videolektioner eller kanske specificera vilka annonser som ska spelas upp före leveransen av den begärda videon. Med detta exempel visas hur du använder ActionScript 3.0 för att göra följande:

- Uppdatera ett spelhuvud baserat på videofilens uppspelningsförlopp
- Avlyssna och tolka videofilens metadata
- Hantera specifika koder i en nätdirektuppspelning
- Läsa in, spela upp, pausa och stoppa en dynamiskt inläst FLV
- Ändra storlek på ett videoobjekt i visningslistan baserat på nätdirektuppspelningens metadata

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Video Jukebox-programfilerna finns i mappen Samples/CapabilitiesExplorer. Programmet består av följande filer:

Fil	Beskrivning
VideoJukebox.fla eller VideoJukebox.mxml	Huvudprogramfilen för Flex (MXML) eller Flash (FLA).
VideoJukebox.as	Klassen som innehåller de flesta funktioner för programmet.
playlist.xml	En fil som visar vilka videofiler som läses in till videojukeboxen.

Läsa in en extern videouppspelningslistfil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den externa playlist.xml-filen specificerar vilka videofiler som ska läsas in och i vilken ordning de ska spelas upp. För att läsa in XML-filen behöver du använda ett URLLoader-objekt och ett URLRequest-objekt enligt följande kod:

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Den här koden placeras i klassen VideoJukebox' konstruktor, så filen läses in innan övrig kod körs. Så fort som XML-filen har lästs in, anropas `xmlCompleteHandler()`-metoden som överför den externa filen till ett XML-objekt enligt följande kod:

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

XML-objektet `playlist` innehåller rå XML-kod från den externa filen, medan XML-videokoden är ett XMLList-objekt som endast innehåller videonoderna. Ett exempel på playlist.xml-filen visas i följande kodutdrag:

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

Slutligen anropar `xmlCompleteHandler()`-metoden `main()`-metoden vilken anger de olika komponentinstanserna i visningslistan såväl som NetConnection- och NetStream-objekten som används för inläsning av de externa FLV-filerna.

Skapa ett användargränssnitt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du måste dra fem Button-instanser till visningslistan och ge dem följande instansnamn för att skapa användargränssnittet: `playButton`, `pauseButton`, `stopButton`, `backButton` och `forwardButton`.

För var och en av de här knappinstanserna måste du tilldela `click`-händelsen en hanterare enligt följande kodutdrag:

```
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);
```

I `buttonClickHandler()`-metoden används en switch-programsats för att fastställa vilka knappinstanser som någon klickar på enligt följande kod:

```
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;
    }
}
```

Sedan lägger du till en Slider-instans i visningslistan och ger den instansnamnet `volumeSlider`. I följande kod sätts skjutreglageinstansens `liveDragging`-egenskap till `true` och en händelseavlyssnare definieras för skjutreglageinstansens `change`-händelse:

```
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);
```

Lägg till en ProgressBar-instans i visningslistan och ge den instansnamnet `positionBar`. Ställ in `mode`-egenskapen på manuell enligt följande kodutdrag:

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Till sist lägger du till en Label-instans i visningslistan och ger den instansnamnet `positionLabel`. Det här Label-instansvärdet anges av timer-instansen.

Avlyssna videoobjektets metadata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När Flash Player påträffar metadata för var och en av de inlästa videofilerna, anropas callback-hanteraren för `onMetaData()` i NetStream-objektets `client`-egenskap. I följande kod initieras ett objekt och den specificerade callback-hanteraren anges:

```
client = new Object();
client.onMetaData = metadataHandler;
```

Med `metadataHandler()`-metoden kopieras dess data till metaegenskapen som definierades tidigare i koden. Det innebär att du har åtkomst till metadata för den aktuella videon när som helst igenom hela programmet. Sedan ändras storleken på videoobjektet på scenen så att den överensstämmer med dimensionerna som returneras från metadatan. Slutligen flyttas förloppsindikatorn för positionBar och storleken ändras baserat på storleken för den video som spelas upp för tillfället. Följande kod innehåller hela `metadataHandler()`-metoden:

```
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;
}
```

Dynamisk inläsning av en video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att läsa in varje video använder programmet ett NetConnection- och NetStream-objekt. I följande kod skapas ett NetConnection-objekt och `null` överförs till `connect()`-metoden. Genom att ange `null`, ansluter Flash Player till en video på den lokala servern i stället för att ansluta till en server som Flash Media Server.

I följande kod skapas både NetConnection- och NetStream-instanserna, en händelseavlyssnare definieras för `netStatus`-händelsen och `client`-egenskapen tilldelas `client`-objektet:

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

Metoden `netStatusHandler()` anropas när status för videon ändras. Det inkluderar start eller stopp av en videouppspelning, buffring eller om en videodirektuppspelning inte hittas. I följande kod visas `netStatusHandler()`-händelsen:

```
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)
    {
        // Ignore any errors.
    }
}
```

Föregående kod utvärderar kodegenskapen för info-objektet och filtrerar fram om koden är ”NetStream.Play.Start”, ”NetStream.Play.StreamNotFound” eller ”NetStream.Play.Stop”. Alla andra koder ignoreras. Om nätdirektuppspelningen startar, startar koden Timer-instansen vilken uppdaterar spelhuvudet. Om nätdirektuppspelningen inte hittas eller stoppas, stoppas Timer-instansen och programmet försöker spela upp nästa video i spelningslistan.

Varje gång Timer körs, uppdaterar `positionBar`-förloppsinsdikatorinstansen sin aktuella position genom att anropa `setProgress()`-metoden för klassen ProgressBar och Label-instansen `positionLabel` uppdateras med den förflutna tiden och total tid för den aktuella videon.

```
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)
    {
        // Ignore this error.
    }
}
```

Styra volymen för videon

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan styra volymen för den dynamiskt inlästa videon genom att ange `soundTransform`-egenskapen i NetStream-objektet. I videojukeboxprogrammet kan du ändra volymnivån genom att ändra värdet för skjutreglageinstansen `volumeSlider`. I följande kod visas hur du kan ändra volymnivån genom att tilldela SoundTransform-objektet värdet från Slider-komponenten vilket anges i `soundTransform`-egenskapen i NetStream-objektet:

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

Styra videouppspelning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Resten av programmet styr videouppspelningen när videon kommer till slutet av videoflödet eller användaren hoppar till föregående eller nästa video.

Med följande metod hämtas video-URL:en från XMLList för den valda positionen:

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

Metoden `playVideo()` anropar `play()`-metoden i NetStream-objektet för att läsa in vald video:

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

Metoden `playPreviousVideo()` minskar aktuell videoindexposition, anropar `playVideo()`-metoden för att läsa in den nya videofilen och anger att förloppsindikatorn ska visas:

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

Den sista metoden `playNextVideo()` ökar videoindexpositionen och anropar `playVideo()`-metoden. Om den aktuella videon är den sista videon i spelningslistan, anropas `clear()`-metoden för Video-objektet och förloppsindikatorinstansens `visible`-egenskap sätts till `false`:

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

Använda klassen StageVideo för maskinvaruaccelererad presentation

Flash Player 10.2 och senare

I Flash Player optimeras videoprestanda med hjälp av maskinvaruacceleration för H.264-avkodning. Du kan ytterligare förbättra prestanda genom att använda StageVideo API. Med StageVideo kan du låta programmet utnyttja fördelarna med maskinvaruaccelererad presentation.

Körningsmiljöer med stöd för StageVideo API omfattar:

- Flash Player 10.2 och senare

Obs! *I Flash Player 11.4/AIR 3.4 och senare kan du använda indata från en kamera tillsammans med StageVideo.*

Du kan hämta källkoden för och mer information om scenvideofunktionen på Komma igång med scenvideo.

En självstudie för snabbstart av scenvideo finns på Arbeta med scenvideo.

Om maskinvaruacceleration med StageVideo

Maskinvaruaccelererad presentation – med videoskalning, färgkonvertering och blitting-teknik – förstärker prestandafördelarna med maskinvaruaccelererad avkodning. På enheter som har GPU-acceleration (maskinvara) kan du använda ett flash.media.StageVideo-objekt för att bearbeta video direkt på enhetens maskinvara. Direktbearbetning frigör CPU-kapacitet för andra uppgifter medan GPU:n hanterar videon. I den äldre Video-klassen används å andra sidan vanligtvis programvarupresentation. Programvarupresentation sker i CPU:n och den kan uppta en avsevärd del av systemresurserna.

För närvarande har få enheter full GPU-acceleration. Med scenvideofunktionen kan dock program maximalt utnyttja den maskinvaruacceleration som trots allt är tillgänglig.

Klassen StageVideo ersätter inte helt klassen Video. Tillsammans ger de här båda klasserna optimal videovisning i förhållande till enhetens resurser vid den aktuella tidpunkten. Ditt program utnyttjar maskinvaruaccelerationen genom att lyssna efter lämpliga händelser och växla mellan StageVideo och Video efter behov.

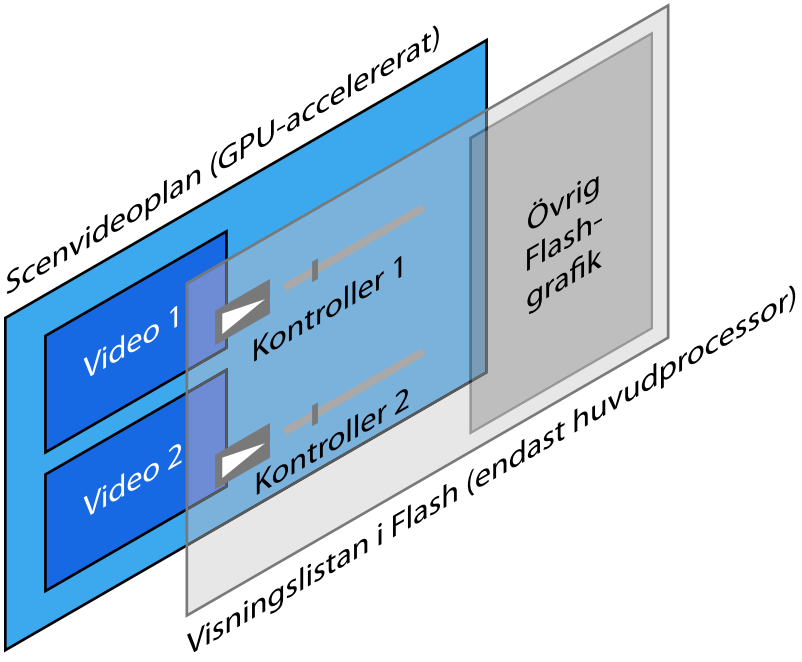
Med klassen StageVideo införs vissa begränsningar på videoanvändning. Innan du implementerar StageVideo bör du granska riktlinjerna och se till att de fungerar i programmet. Om du accepterar begränsningarna kan du använda klassen StageVideo när maskinvaruaccelererad presentation identifieras i Flash Player. Se ”Riktlinjer och begränsningar” på sidan 514.

Parallella planer: Scenvideo och Flash-visningslistan

Med scenvideomodellen kan separat video spelas upp i Flash Player från visningslistan. Den sammansatta visningen delas upp mellan två z-ordnade plan i Flash Player:

Scenvideoplan Scenvideoplanet ligger i bakgrunden. Där visas endast maskinvaruaccelererad video. På grund av den här designen är det här planet inte tillgängligt om maskinvaruacceleration inte stöds eller inte är tillgängligt på enheten. I ActionScript hanterar StageVideo-objekt video som spelas upp på scenvideoplanet.

Plan med visningslistan i Flash Enheter på visningslistan i Flash sätts samman i ett plan framför scenvideoplanet. Entiteter av visningslistor omfattar allt som återges i körningstiden, inklusive uppspelningskontroller. När maskinvaruacceleration inte är tillgängligt kan videofilmer bara spelas upp på det här planet med Video-klassobjektet. Scenvideo visas alltid bakom grafiken i visningslistan i Flash.



Videovisningsplan

StageVideo-objektet visas i en icke-roterad, rektangulär del av skärmen, justerad efter fönstret. Du kan inte lägga objekt i lager bakom scenvideoplanet. Du kan emellertid använda planet med visningslistan i Flash för att placera andra bilder i lager ovanpå scenvideoplanet. Scenvideo körs samtidigt med visningslistan i På så sätt kan du använda de två funktionerna tillsammans för att skapa en visuellt enhetlig effekt med två separata plan. Du kan till exempel använda det främre planet för uppspelningskontroller, som används på scenvideon i bakgrunden.

Scenvideo och H.264-kodek

I Flash Player-program innebär implementering av maskinvaruacceleration för video två steg:

1 Kodning av videon som H.264

2 Implementering av StageVideo API

För bästa resultat ska du utföra båda stegen. Med H.264-kodeken kan du till fullo utnyttja maskinvaruacceleration, från videoavkodning till presentation.

Scenvideo eliminerar GPU-till-CPU-återläsning. GPU skickar alltså inte längre avkodade bildrutor tillbaka till CPU:n för sammansättning med visningslisteobjekt. I stället skickar GPU:n avkodade och återgivna bildrutor direkt till skärmen, bakom visningslisteobjekten. Tekniken minskar CPU- och minnesanvändning samt förbättrar pixelåtergivningen.

Fler hjälpavsnitt

”Videoformat” på sidan 475

Riktlinjer och begränsningar

När video körs i helskärmsläge är scenvideo alltid tillgängligt om enheten har stöd för maskinvaruacceleration. Flash Player körs dessutom i en webbläsare. I webbläsarsammanhang kan inställningen `wmode` påverka scenvideons tillgänglighet. Därför bör du alltid använda `wmode="direct"` om du vill att scenvideo ska vara tillgängligt. Scenvideo är inte kompatibelt med andra `wmode`-inställningar om inte helskärmsläge används. Den här begränsningen innebär att scenvideo kan växla mellan att vara tillgängligt och otillgängligt på ett oförutsägbart sätt under körning. Om användaren till exempel avslutar helskärmsläget när scenvideon spelas upp, ändras videosammanhanget tillbaka till webbläsaren. Om parametern `wmode` för webbläsaren inte har angetts som `"direct"` kan scenvideon plötsligt bli otillgänglig. Information om ändringar av uppspelningssammanhanget skickas från Flash Player till program genom en uppsättning händelser. Om du implementerar StageVideo API ska du behålla ett Video-objekt som säkerhetsnivå när scenvideo blir otillgängligt.

På grund av scenvideons direkta förhållande till maskinvaran begränsar den vissa videofunktioner. Scenvideo påtvingar följande begränsningar:

- För varje SWF-fil begränsas antalet StageVideo-objekt som samtidigt kan visa video till fyra i Flash Player. Den faktiska gränsen kan vara ännu lägre, beroende på enhetens maskinvaruresurser.
- Videotimingen är inte synkroniserad med timingen av innehåll som visas i körningsmiljön.
- Visningsområdet för videofilmen kan bara vara en rektangel. Du kan inte använda mer avancerade visningsområden, som elliptiska eller oregelbundna former.
- Du kan inte rotera videofilmen.
- Du kan inte bitmappscacha videon eller använda BitmapData-objekt för att komma åt den.
- Du kan inte använda filter på videofilmen.
- Du kan inte använda färgomvandlingar på videofilmen.
- Du kan inte använda ett alfavärde på videofilmen.
- Blandningslägen som du tillämpar på objekt i visningslisterutan gäller inte scenvideo.
- Du kan bara placera videofilmen på helpixelgränser.
- Även om GPU-återgivningen är den bästa möjliga för aktuell enhetsmaskinvara är den inte ”pixelidentisk” till 100 % på alla enheter. Mindre variationer förekommer beroende på drivrutin och plattform.
- Ett fåtal enheter saknar stöd för alla färgrymder som krävs. Vissa enheter har till exempel inte stöd för BT.709, H.264-standarden. I så fall kan du använda BT.601 för snabb visning.
- Du kan inte använda scenvideo med WMODE-inställningar som normal, ogenomskinlig eller genomskinlig. Scenvideo har bara stöd för `WMODE=direct` om inte helskärmsläge används. WMODE har ingen effekt i Safari 4 eller senare, och i IE 9 eller senare.

I de flesta fall påverkar de här begränsningarna inte videouppspelningsprogram. Om du kan acceptera begränsningarna bör du alltid använda scenvideo om det går.

Fler hjälpavsnitt

”Arbeta i helskärmsläge” på sidan 167

Använda StageVideo-API:erna

Scenvideo är en funktion i körningstiden som förbättrar videouppspelning och enhetsprestanda. Funktionen skapas och bibehålls av körningstiden. Som utvecklare består ditt arbete av att konfigurera ditt program så att det kan utnyttja funktionen.

För att utnyttja scenvideo implementerar du ett ramverk med händelsehanterare, som identifierar när scenvideo är tillgängligt och inte. När du får ett meddelande om att scenvideo är tillgängligt hämtar du ett StageVideo-objekt från egenskapen `Stage.stageVideos`. Det här Vector-objektet fylls automatiskt i med ett eller flera StageVideo-objekt. Du kan sedan använda ett av dessa StageVideo-objekt, i stället för ett Video-objekt, för att visa direktuppspelad video.

När du får ett meddelande i Flash Player om att scenvideo inte längre är tillgängligt växlar du tillbaka videoströmmen till ett Video-objekt.

Obs! *Du kan inte skapa StageVideo-objekt.*

Egenskapen Stage.stageVideos

Egenskapen `Stage.stageVideos` är ett Vector-objekt som ger dig tillgång till StageVideo-instanser. Den här vektorn kan innehålla upp till fyra StageVideo-objekt, beroende på maskinvara och systemresurser. Mobilenheter kan vara begränsade till ett eller inga alls.

När scenvideo inte är tillgängligt innehåller den här vektorn inga objekt. Se till att du bara använder medlemmar i den här vektorn när den senaste `StageVideoAvailability`-händelsen anger att scenen är tillgänglig, annars kan du råka ut för körningsfel.

StageVideo-händelser

Ramverket för StageVideo-API:t tillhandahåller följande händelser:

StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY Skickas när egenskapen `Stage.stageVideos` ändras. Egenskapen `StageVideoAvailabilityEvent.availability` anger antingen `AVAILABLE` eller `UNAVAILABLE`. Använd den här händelsen för att avgöra om egenskapen `stageVideos` innehåller några StageVideo-objekt, i stället för att kontrollera längden på `Stage.stageVideos`-vektorn direkt.

StageVideoEvent.RENDER_STATE Skickas när ett NetStream- eller Camera-objekt har kopplats till ett StageVideo-objekt och håller på att spelas upp. Anger vilken typ av avkodning som används: maskinvara, programvara eller ej tillgängligt (inget visas). Händelsemålet innehåller egenskaperna `videoWidth` och `videoHeight` som utan risk kan användas för att ändra storlek på videovisningsrutan.

Viktigt! *Koordinater som hämtas från StageVideo-målobjektet använder Stage-koordinater eftersom de inte ingår i standardvisningslistan.*

VideoEvent.RENDER_STATE Skickas när ett Video-objekt används. Anger om programvaruavkodning eller maskinvaruaccelererad avkodning används. Om den här händelsen visar på maskinvaruaccelererad avkodning växlar du om möjligt till ett StageVideo-objekt. Video-händelsemålet innehåller egenskaperna `videoWidth` och `videoHeight` som utan risk kan användas för att ändra storlek på videovisningsrutan.

Arbetsflöde för att implementera funktionen StageVideo

Följ de här stegen om du vill implementera funktionen StageVideo:

1. Lyssna efter händelsen `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`om du vill veta när vektorn `Stage.stageVideos` har ändrats. Mer information finns i ”Använda händelsen StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY” på sidan 517.
2. Om händelsen `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` rapporterar att scenvideo är tillgängligt använder du vektorobjektet `Stage.stageVideos` i händelsehanteraren för att få åtkomst till ett StageVideo-objekt.
3. Bifoga ett NetStream-objekt med `StageVideo.attachNetStream()` eller bifoga ett Camera-objekt med `StageVideo.attachCamera()`.
4. Spela upp videofilmen med `NetStream.play()`.
5. Lyssna efter händelsen `StageVideoEvent.RENDER_STATE` på StageVideo-objektet om du vill fastställa statusen för den video som spelas upp. Ett kvitto på den här händelsen anger även att bredd- och höjdegenskaperna för den videon har initierats eller ändrats. Mer information finns i ”Använda händelserna StageVideoEvent.RENDER_STATE och VideoEvent.RENDER_STATE” på sidan 519.
6. Lyssna efter händelsen `VideoEvent.RENDER_STATE` på Video-objektet. Den här händelsen tillhandahåller samma status som `StageVideoEvent.RENDER_STATE`, så du kan använda den om du vill fastställa om GPU-acceleration är tillgänglig. Ett kvitto på den här händelsen anger även att bredd- och höjdegenskaperna för den videon har initierats eller ändrats. Mer information finns i ”Använda händelserna StageVideoEvent.RENDER_STATE och VideoEvent.RENDER_STATE” på sidan 519.

Initiera StageVideo-händelseavlyssnare

Konfigurera StageVideoAvailabilityEvent- och VideoEvent-avlyssnare under programinitieringen. Du kan till exempel initiera dessa lyssnare i `flash.events.Event.ADDED_TO_STAGE`-händelsehanteraren. Den här händelsen garanterar att programmet syns på scenen:

```
public class SimpleStageVideo extends Sprite
    private var nc:NetConnection;
    private var ns:NetStream;

    public function SimpleStageVideo()
    {
        // Constructor for SimpleStageVideo class
        // Make sure the app is visible and stage available
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }

    private function onAddedToStage(event:Event):void
    {
        //...
        // Connections
        nc = new NetConnection();
        nc.connect(null);
        ns = new NetStream(nc);
        ns.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
        ns.client = this;

        // Screen
        video = new Video();
        video.smoothing = true;

        // Video Events
        // the StageVideoEvent.STAGE_VIDEO_STATE informs you whether
        // StageVideo is available
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
            onStageVideoState);
        // in case of fallback to Video, listen to the VideoEvent.RENDER_STATE
        // event to handle resize properly and know about the acceleration mode running
        video.addEventListener(VideoEvent.RENDER_STATE, videoStateChange);
        //...
    }
```

Använda händelsen StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY

I `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`-hanteraren avgör du om du ska använda ett Video- eller ett StageVideo-objekt baserat på tillgängligheten för StageVideo. Om egenskapen `StageVideoAvailabilityEvent.availability` är inställd på `StageVideoAvailability.AVAILABLE` använder du StageVideo. I det här fallet kan du vara säker på att Stage.stageVideos-vektorn innehåller ett eller flera StageVideo-objekt. Hämta ett StageVideo-objekt från egenskapen `Stage.stageVideos` och bifoga NetStream-objektet till det. Eftersom StageVideo-objekt alltid visas i bakgrunden tar du bort eventuella Video-objekt (alltid i förgrunden). Du använder även den här händelsehanteraren för att lägga till en avlyssnare för `StageVideoEvent.RENDER_STATE`-händelsen.

Om egenskapen `StageVideoAvailabilityEvent.availability` är inställd på `StageVideoAvailability.UNAVAILABLE` använder du inte StageVideo och öppnar inte heller `Stage.stageVideos`-vektorn. Bifoga i så fall NetStream-objektet till ett Video-objekt. Lägg till slut till StageVideo- eller Video-objektet till scenen och anropa `NetStream.play()`.

Följande kod visar hur händelsen `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` ska hanteras:

```
private var sv:StageVideo;
private var video:Video;

private function onStageVideoState(event:StageVideoAvailabilityEvent):void
{
    // Detect if StageVideo is available and decide what to do in toggleStageVideo
    toggleStageVideo(event.availability == StageVideoAvailability.AVAILABLE);
}

private function toggleStageVideo(on:Boolean):void
{
    // To choose StageVideo attach the NetStream to StageVideo
    if (on)
    {
        stageVideoInUse = true;
        if ( sv == null )
        {
            sv = stage.stageVideos[0];
            sv.addEventListener(StageVideoEvent.RENDER_STATE, stageVideoStateChange);
                sv.attachNetStream(ns);
        }

        if (classicVideoInUse)
        {
            // If you use StageVideo, remove from the display list the
            // Video object to avoid covering the StageVideo object
            // (which is always in the background)
            stage.removeChild ( video );
            classicVideoInUse = false;
        }
    } else
    {
        // Otherwise attach it to a Video object
        if (stageVideoInUse)
            stageVideoInUse = false;
        classicVideoInUse = true;
        video.attachNetStream(ns);
        stage.addChildAt(video, 0);
    }

    if ( !played )
    {
        played = true;
        ns.play(FILE_NAME);
    }
}
```

Viktigt! *Den första gången ett program öppnar vektorelementet vid Stage.stageVideos[0] anges standardrektangeln till 0,0,0,0, och för panorerings- och zoomegenskaperna används standardvärden. Återställ alltid de här värdena som du vill ha dem. Du kan använda egenskaperna* `videoWidth` *och* `videoHeight` *för händelsemålet* `StageVideoEvent.RENDER_STATE` *eller* `VideoEvent.RENDER_STATE` *för att beräkna videovisningsrutans dimensioner.*

Hämta den fullständiga källkoden för det här exempelprogrammet på Komma igång med scenvideo.

Använda händelserna StageVideoEvent.RENDER_STATE och VideoEvent.RENDER_STATE

StageVideo- och Video-objekt skickar händelser som informerar program när återgivningsmiljön visas. Händelserna är `StageVideoEvent.RENDER_STATE` och `VideoEvent.RENDER_STATE`.

Ett StageVideo- eller Video-objekt skickar en återgivningstillståndshändelse när ett NetStream-objekt bifogas och börjar spelas upp. Dessutom skickas den här händelsen när återgivningsmiljön ändras, till exempel om videovisningsrutan får en ny storlek. Använd dessa aviseringar för att återställa visningsrutan till aktuella `videoHeight`- och `videoWidth`-värden för händelsemålobjektet.

Rapporterade återgivningstillstånd omfattar:

- `RENDER_STATUS_UNAVAILABLE`
- `RENDER_STATUS_SOFTWARE`
- `RENDER_STATUS_ACCELERATED`

Återgivningstillstånden anger när maskinvaruaccelererad avkodning används, oavsett vilken klass som för närvarande spelar upp video. Kontrollera egenskapen `StageVideoEvent.status` om du vill ta reda på om den nödvändiga avkodningen är tillgänglig. Om egenskapen anges som "unavailable" kan StageVideo-objektet inte spela upp videon. Vid det här tillståndet krävs det att du omedelbart bifogar NetStream-objektet igen till ett Video-objekt. Andra tillstånd informerar programmet om aktuella återgivningsförhållanden.

I följande tabell beskrivs innebörden av alla värden för återgivningstillstånd för StageVideoEvent- och VideoEvent-objekt i Flash Player:

	VideoStatus.ACCELERATED	**VideoStatus.SOFTWARE**	**VideoStatus.UNAVAILABLE**
StageVideoEvent	Både avkodning och presentation sker i maskinvaran. (Bästa prestanda.)	Presentation i maskinvara, avkodning i programvara. (Acceptabla prestanda.)	Inga GPU-resurser är tillgängliga för att hantera videon och inget visas. **Fall tillbaka på ett Video-objekt.**
VideoEvent	Presentation i programvara, avkodning i maskinvara. (Acceptabla prestanda endast på modern, stationär dator. Försämrade helskärmsprestanda.)	Presentation i programvara, avkodning i programvara. (Sämst för prestanda. Försämrade helskärmsprestanda.)	(Ej tillämpligt)

Färgrymder

Scenvideo använder underliggande maskinvarufunktioner för att ge stöd åt färgrymder. SWF-innehåll kan tillhandahålla metadata som anger önskad färgrymd. Det är emellertid enhetens maskinvara för grafik som avgör om den färgrymden kan användas. Vissa enheter har stöd för flera färgrymder, andra inga alls. Om maskinvaran saknar stöd för den begärda färgrymden försöker Flash Player hitta den bästa matchningen bland de färgrymder som stöds.

Om du vill veta vilka färgrymder maskinvaran har stöd för använder du egenskapen `StageVideo.colorSpaces`. Den här egenskapen returnerar en lista över de färgrymder som stöds i en String-vektor:

```
var colorSpace:Vector.<String> = stageVideo.colorSpaces();
```

Om du vill veta vilken färgrymd den video som spelas upp använder kontrollerar du egenskapen `StageVideoEvent.colorSpace`. Kontrollera den här egenskapen i händelsehanteraren för händelsen `StageVideoEvent.RENDER_STATE`:

```
var currColorSpace:String;

//StageVideoEvent.RENDER_STATE event handler
private function stageVideoRenderState(event:Object):void
{
    //...
    currColorSpace = (event as StageVideoEvent).colorSpace;
    //...
}
```

Om det inte går att hitta en ersättning för en färgrymd som inte stöds använder scenvideon standardfärgrymden BT.601. Videoströmmar med H.264-kodning använder till exempel oftast BT.709-färgrymden. Om enhetens maskinvara saknar stöd för BT.709 returnerar egenskapen `colorSpace` värdet `"BT601"`. Ett `StageVideoEvent.colorSpace`-värde på `"unknown"` anger att maskinvaran saknar funktioner för att fråga efter färgrymden.

Om den aktuella färgrymden inte kan användas av ditt program kan du välja att växla från ett StageVideo-objekt till ett Video-objekt. Klassen Video har stöd för alla färgrymder via programvarusammansättning.

Kapitel 26: Arbeta med kameror

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En kamera som är ansluten till användarens dator kan fungera som källa för videodata som du kan visa och ändra med ActionScript. Klassen Camera är inbyggd i ActionScript och används för att arbeta med en dators eller enhets kamera.

På mobilenheter kan du också använda klassen CameraUI. Klassen CameraUI startar ett separat kameraprogram så att användaren kan hämta en stillbild eller en videofilm. När användaren är klar kan ditt program öppna bilden eller videon via ett MediaPromise-objekt.

Fler hjälpavsnitt

Christian Cantrell: How to use CameraUI in a Cross-platform Way (Använda CameraUI på flera plattformar)

Michaël CHAIZE: Android, AIR and the Camera (Android, AIR och kameror)

Klassen Camera

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med Camera-objektet kan du ansluta användarens lokala kamera och sända videon lokalt (tillbaka till användaren) eller externt till en server (t.ex Flash Media Server).

När du använder Camera-klassen har du åtkomst till följande information om användarens kamera:

- Vilka kameror som installerats på och är tillgängliga för användarens dator eller enhet
- Om en kamera har installerats
- Om Flash Player kan neka åtkomst till användarens kamera
- Vilken kamera som är aktiv
- Bredden och höjden för videon som spelas in.

Klassen Camera innehåller flera användbara metoder och egenskaper för arbete med kameraobjekt. Den statiska `Camera.names`-egenskapen innehåller t.ex. en array med kameranamn som finns installerade på användarens dator. Du kan också använda `name`-egenskapen för att visa namnet på den kamera som är aktiv.

Obs! *När video från kameran direktuppspelas över nätverket bör du alltid hantera nätverksavbrott. Nätverksavbrott kan ske av många olika skäl, särskilt på mobilenheter.*

Visa kamerainnehåll på skärmen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du ansluter till en kamera kan det kräva mindre kod än när du använder klasserna NetConnection och NetStream för att läsa in en videofil. Kameraklassen kan också snart bli svårhanterlig eftersom du behöver användarens tillstånd att ansluta Flash Player till kameran innan du får åtkomst till den.

I koden nedan visas hur du kan använda klassen Camera för att ansluta till användarens lokala kamera:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Obs! *Klassen Camera har ingen konstruktormetod. När du vill skapa en ny Camera-instans använder du den statiska* `Camera.getCamera()`*-metoden.*

Utforma kameraprogram

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skriver ett program som ansluter till användarens kamera behöver du ta hänsyn till följande i koden:

- Kontrollera om användaren har installerat en kamera. Hantera de fall då ingen kamera är tillgänglig.
- När det gäller Flash Player måste du kontrollera om användaren uttryckligen har tillåtit åtkomst till kameran. Av säkerhetsskäl visas dialogrutan Flash Player-inställningar i spelaren där användaren kan tillåta eller neka åtkomst till kameran. Det förhindrar att Flash Player ansluter till en användares kamera och sänder ett videoflöde utan tillstånd. Om användaren tillåter det, kan ditt program ansluta till användarens kamera. Om användaren nekar det, kan ditt program inte ansluta till användarens kamera. Ditt program bör alltid hantera båda fallen på ett smidigt sätt.
- (Gäller endast AIR) Kontrollera om klassen Camera stöds för de enhetsprofiler som programmet har stöd för.
- Klassen Camera stöds inte i mobilwebbläsare.
- Klassen Camera stöds inte i AIR-program för mobiler som använder GPU-återgivning.
- På mobilenheter kan endast en kamera i taget vara aktiv.

Fler hjälpavsnitt

Christophe Coenraets: Multi-User Video Tic-Tac-Toe (Luffarschack för flera användare)

Mark Doherty: Android Radar app (source) (Radar för Android (källkod))

Ansluta till användarens kamera

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Steg ett när du ansluter till en användares kamera är att skapa en ny kamerainstans genom att skapa en variabel av typen Camera och initiera den att returnera värden från den statiska `Camera.getCamera()`-metoden.

I nästa steg skapar du ett nytt videoobjekt och kopplar Camera-objektet till det.

I steg tre lägger du till videoobjektet i visningslistan. Du måste utföra steg 2 och 3 eftersom klassen Camera inte utökar klassen DisplayObject, vilket betyder att den inte kan läggas till direkt i visningslistan. För att visa den kamerainspelade videon skapar du ett nytt videoobjekt och anropar `attachCamera()`-metoden.

De tre stegen visas i följande kod:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Observera, att om användaren inte har installerat någon kamera visar programmet ingenting.

I verkligheten måste du utföra ytterligare steg i programmet. Mer information finns i ”Verifiera att en kamera har installerats” på sidan 523 och ”Identifiera tillstånd för kameraåtkomst” på sidan 524.

Fler hjälpavsnitt

Lee Brimelow: How to access the camera on Android devices (Komma åt kameran på Android-enheter)

Verifiera att en kamera har installerats

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Innan du försöker använda några metoder eller egenskaper på en kamerainstans kontrollerar du att användaren har installerat en kamera. Det finns två sätt att kontrollera om användaren har en kamera installerad:

- Kontrollera den statiska `Camera.names`-egenskapen som innehåller en array med kameranamn som är tillgängliga. Vanligtvis har arrayen endast en eller inga strängar alls eftersom de flesta användare troligen inte har fler än en kamera installerad samtidigt. I följande kod visas hur du kan kontrollera `Camera.names`-egenskapen för att se om användaren har en tillgänglig kamera:

  ```
  if (Camera.names.length > 0)
  {
      trace("User has at least one camera installed.");
      var cam:Camera = Camera.getCamera(); // Get default camera.
  }
  else
  {
      trace("User has no cameras installed.");
  }
  ```

- Kontrollera returvärdet för den statiska `Camera.getCamera()`-metoden. Om det inte finns några tillgängliga kameror returnerar metoden `null`, annars returneras en referens till ett Camera-objekt. I följande kod visas hur du kan kontrollera `Camera.getCamera()`-metoden för att se om användaren har en tillgänglig kamera:

  ```
  var cam:Camera = Camera.getCamera();
  if (cam == null)
  {
      trace("User has no cameras installed.");
  }
  else
  {
      trace("User has at least 1 camera installed.");
  }
  ```

Eftersom klassen Camera inte utökar klassen DisplayObject, kan den inte läggas till direkt i visningslistan med metoden `addChild()`. För att visa den kamerainspelade videon måste du skapa ett nytt videoobjekt och anropa `attachCamera()`-metoden i Video-instansen.

I kodutdraget visas hur du kan bifoga en kamera om det finns någon. Om inte, visar programmet ingenting:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

Kameror på mobilenheter

Klassen Camera stöds inte i Flash Player-miljön i mobilwebbläsare.

I AIR-program på mobilenheter kan du komma åt kameran eller kamerorna på enheten. I mobila enheter kan du använda både den framåt- och bakåtriktade kameran, men det går bara att visa utdata från en kamera i taget. (Om du ansluter en andra kamera kopplas den första bort.) Den framåtriktade kameran speglas horisontellt i iOS, men inte i Android.

Identifiera tillstånd för kameraåtkomst

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I AIR-programmets säkerhetssandlåda kan programmet få tillgång till alla kameror utan användarens tillåtelse. På Android måste programmet däremot ange Android-behörigheten CAMERA i programbeskrivningsfilen.

Innan Flash Player kan visa en kameras utdata måste användaren uttryckligen tillåta Flash Player att få åtkomst till kameran. När `attachCamera()`-metoden anropas, visas dialogrutan Flash Player-inställningar som uppmanar användaren att antingen tillåta eller neka Flash Player åtkomst till kameran och mikrofonen. Om användaren klickar på Tillåt, visas kamerainspelningen i Video-instansen på scenen. Om användaren klickar på Neka, kan inte Flash Player ansluta till kameran och videoobjektet visar ingenting.

Om du vill ta reda på om användaren tillåter åtkomst till kameran, kan du avlyssna kamerans `status`-händelse (`StatusEvent.STATUS`), enligt exempel i följande kod:

```
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
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

Funktionen `statusHandler()` anropas så fort som användaren klickar på Tillåt eller Neka. Du kan ta reda på vilken knapp användaren klickade på med någon av de här två metoderna:

- `event`-parametern för `statusHandler()`-funktionen innehåller en kodegenskap som innehåller strängen ”Camera.Muted” eller ”Camera.Unmuted”. Om värdet är ”Camera.Muted” klickade användaren på Neka och Flash Player får inte åtkomst till kameran. Ett exempel på det här visas i följande utdrag:

```
function statusHandler(event:StatusEvent):void
{
    switch (event.code)
    {
        case "Camera.Muted":
            trace("User clicked Deny.");
            break;
        case "Camera.Unmuted":
            trace("User clicked Accept.");
            break;
    }
}
```

- Klassen Camera innehåller en skrivskyddad egenskap med namnet `muted` som specificerar om användaren nekar åtkomst till kameran (`true`) eller tillåter åtkomst (`false`) på panelen Sekretess i Flash Player. Ett exempel på det här visas i följande utdrag:

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

Genom att kontrollera när statushändelsen ska skickas, kan du skriva kod som hanterar när användaren tillåter eller nekar åtkomst till kameran och rensa upp på rätt sätt. Om användaren t.ex. klickar på Neka kan du visa ett meddelande om att avändaren måste klicka på Tillåt om han eller hon vill delta i en videochattsession, eller så kan du i stället kontrollera att videoobjektet i visningslistan tas bort för att frigöra systemresurser.

I AIR skickar ett Camera-objekt inga statushändelser, eftersom behörigheten att använda kameran inte är dynamisk.

Maximera kamerans videokvalitet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Som standard är nya instanser i Video-klassen 320 pixlar breda och 240 pixlar höga. För att maximera videokvaliteten bör du alltid kontrollera att videoobjektet har samma dimensioner som videon som returneras från kameraobjektet. Du kan hämta kamerans bredd och höjd med klassen Cameras `width`- och `height`-egenskaper, sedan kan du ange videoobjektets `width`- och `height`-egenskaper så att de stämmer överens med kameraobjektets dimensioner, eller också kan du överföra kamerans bredd och höjd till klassen Videos konstruktormetod enligt exempel i följande kodutdrag:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Eftersom `getCamera()`-metoden returnerar en referens till kameraobjektet (eller `null` om ingen kamera är tillgänglig) har du åtkomst till kamerans metoder och egenskaper även om användaren nekar åtkomst till sin kamera. Det här innebär att du kan ange storleken för videoinstansen med kamerans ursprungliga höjd och bredd.

```
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
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Mer information om helskärmsläget finns i avsnittet om helskärmsläge under ”Inställning av scenegenskaper” på sidan 164.

Övervaka kamerans status

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Camera innehåller flera egenskaper med vilka du kan övervaka Camera-objektets aktuella status. I följande kod visas t.ex. flera av kameraegenskaperna som använder ett Timer-objekt och en textfältsinstans i visningslistan:

```
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");
}
```

Var 1/10 sekund (100 millisekunder) skickas Timer-objektets `timer`-händelse och `timerHandler()`-funktionen uppdaterar textfältet i visningslistan.

Kapitel 27: Använda DRM (Digital Rights Management)

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

Adobe® Access™ är en lösning för innehållsskydd. Det gör det möjligt för innehållsägare, distributörer och annonsörer att skapa nya inkomstkällor med hjälp av en sömlös åtkomst till betalinnehåll. Med Adobe Access kan utgivaren kryptera innehåll, skapa principer och utfärda licenser. Adobe Flash Player och Adobe AIR innehåller ett DRM-bibliotek, Adobe Access-modulen. Med den här modulen kan körningen kommunicera med en Adobe Access-licensserver och spela upp skyddat innehåll. Körningen blir därmed den sista och fullgörande fasen av livscykeln för innehåll som skyddas av Adobe Access och distribueras med Flash Media Server.

Med Adobe Access kan innehållsleverantörer erbjuda både gratis innehåll och betalt innehåll. Ett exempel kan vara en konsument som vill se ett tv-program utan reklamavbrott. Konsumenten registrerar sig och betalar innehållsutgivaren. Konsumenten kan sedan med hjälp av sina inloggningsuppgifter få åtkomst till och spela upp programmet utan reklam.

Ett annat exempel är en konsument som vill se innehållet offline när han befinner sig på en resa och inte har någon Internetåtkomst. Offline-arbetsflöden som dessa stöds i AIR-programmen. När användaren har registrerat sig och betalat utgivaren för innehållet kan denne få åtkomst till och ladda ner innehållet och tillhörande AIR-program från utgivarens webbplats. Användaren kan sedan visa innehållet offline under den tillåtna tidsperioden med hjälp av AIR-programmet. Innehållet kan även delas med andra enheter i samma enhetsgrupp med hjälp av domäner (**Nytt i 3.0**).

Adobe Access har även stöd för anonym åtkomst som inte kräver någon användarautentisering. En utgivare kan till exempel använda anonym åtkomst för att distribuera annonsfinansierat innehåll. Via anonym åtkomst kan utgivaren även ge gratis tillgång till det aktuella innehållet under ett visst antal dagar. Innehållsleverantören kan också ange och begränsa vilken typ och version av spelare som behövs för innehållet.

När en användare vill spela upp skyddat innehåll i Flash Player eller Adobe AIR måste programmet anropa API:erna för DRM. API:erna för DRM initierar arbetsflödet för uppspelning av det skyddade innehållet. Genom Adobe Access-modulen kontaktas licensservern under körningen. Användaren autentiseras om nödvändigt av licensservern och en licens utfärdas för att tillåta uppspelningen av skyddat innehåll. Körningen tar emot licensen och dekrypterar innehållet för uppspelning.

Här beskrivs hur du gör för att spela upp innehåll som skyddas av Adobe Access med ditt program. Du behöver inte förstå hur du krypterar innehåll eller upprätthåller principer med Adobe Access. Däremot måste du kommunicera med Adobe Access-licensservern för att autentisera användaren och hämta licensen. Du måste också utforma ett program särskilt för uppspelning av innehåll som skyddas av Adobe Access.

En översikt över Flash Access, med information om hur du skapar principer, finns i dokumentationen som ingår i Adobe Access.

Fler hjälpavsnitt

flash.net.drm-paketet

flash.net.NetConnection

flash.net.NetStream

Förstå arbetsflödet för skyddat innehåll

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Viktigt: I Flash Player 11.5 och senare integreras Adobe Access-modulen så uppdateringssteget (anrop av `SystemUpdater.update(SystemUpdaterType.DRM)`) är inte nödvändigt. Detta gäller följande webbläsare och plattformar:

- ActiveX-kontoll för Flash Player 11.5, för alla plattformar utom Internet Explorer i Windows 8 på Intel-processorer
- Plugin-program för Flash Player 11.5, för alla webbläsare
- Adobe AIR (dator och mobil)

Detta innebär att uppdateringssteget *fortfarande krävs* i följande fall:

- Internet Explorer i Windows 8 på Intel-processorer
- Flash Player 11.4 och senare, utom Google Chrome 22 och senare (alla plattformar) eller 21 och senare (Windows)

Obs! *Det är fortfarande säkert att anropa* `SystemUpdater.update(SystemUpdaterType.DRM)` *i ett system med Flash Player 11.5 eller senare, men inget laddas ned.*

Följande arbetsflöde på hög nivå beskriver hur ett program kan hämta och spela upp skyddat innehåll. För det här arbetsflödet förutsätts det att programmet utformas särskilt för uppspelning av innehåll som skyddas av Adobe Access:

1. Hämta innehållets metadata.
2. Hantera uppdateringar av Flash Player, vid behov.
3. Kontrollera om det finns en licens lokalt tillgänglig. Om det finns en licens läser du in den och går vidare till steg 7. Om det inte gör det går du till steg 4.
4. Kontrollera om autentisering krävs. Om det inte gör det går du vidare till steg 7.
5. Om autentisering krävs måste inloggningsuppgifterna hämtas från användaren och skickas till licensservern.
6. Om en domänregistrering krävs ska du gå med i domänen (AIR 3.0 och senare).
7. När autentiseringen har slutförts kan licensen hämtas från servern.
8. Spela upp innehållet.

Om inget fel inträffar och användaren autentiserades för att se innehållet, skickar NetStream-objektet ett DRMStatusEvent-objekt. Programmet påbörjar då uppspelningen. DRMStatusEvent-objektet innehåller relaterad voucherinformation, som identifierar användarens policy och behörigheter. Till exempel innehåller det information om innehållet kan göras tillgängligt offline eller när licensen upphör. Programmet kan använda data för att informera användaren om statusen för policyn. Programmet kan till exempel visa i statusfältet hur många dagar till som användaren kan visa innehållet.

Om användaren tillåts åtkomst offline, cache-lagras vouchern och det krypterade innehållet laddas ner till användarens dator. Innehållet är tillgängligt under den tid som angivits för licensens varaktighet i cache-minnet. Egenskapen `detail i händelsen innehåller "DRM.voucherObtained"`. Programmet avgör var innehållet ska sparas lokalt för att kunna visas offline. Du kan även förhandsladda en voucher med klassen DRMManager.

Obs! *Cache-lagring och inläsning av vouchers i förväg stöds i både AIR och Flash Player. Hämtning och förvaring av krypterat innehåll stöds endast i AIR.*

I programmet hanteras alla felhändelser explicit. Dessa händelser omfattar fall där användaren anger giltiga inloggningsuppgifter, men där åtkomsten till innehållet begränsas av vouchern som skyddar det krypterade innehållet. En autentiserad användare kan till exempel inte få åtkomst till innehållet om denne inte har betalat för rättigheterna. Detta kan också inträffa när två användare, som båda är registrerade medlemmar hos samma utgivare, försöker att dela innehåll som bara en av dem har betalat för. Programmet ska informera användaren om felet och presentera ett alternativ. Ett vanligt alternativ är att ge användaren information om hur han/hon registrerar sig och betalar för att få visa innehåll.

Utförligt API-arbetsflöde

Flash Player 10.1 och senare, AIR 2.0 och senare

Det här arbetsflödet ger en mer utförlig beskrivning av arbetsflödet för skyddat innehåll. Det här arbetsflödet beskriver de specifika API:er som används för att spela upp innehåll som skyddas av Adobe Access.

1. Läs in de byte som hör till filen för det skyddade innehållets metadata med ett URLLoader-objekt. Ange en variabel för objektet, till exempel `metadata_bytes`.

   Allt innehåll som kontrolleras av Adobe Access har Adobe Access-metadata. När innehållet packas kan dessa metadata sparas som en separat metadata-fil (.metadata) tillsammans med innehållet. Mer information finns i dokumentationen för Adobe Access.

2. Skapa en DRMContentData-instans. Lägg den här koden i ett try-catch-block:

   ```
   new DRMContentData(metadata_bytes)
   ```

   där *`metadata_bytes`* är URLLoader-objektet som fås i steg 1.

3. (Endast Flash Player) Körningen kontrollerar att Adobe Access-modulen finns. Om den inte hittas inträffar ett IllegalOperationError-fel med felkoden 3344 eller 3343 för DRMErrorEvent.

   Du hantera felet genom att hämta Adobe Access-modulen med API:n SystemUpdater. När modulen har hämtats skickar SystemUpdater-objektet en COMPLETE-händelse. Inkludera en händelseavlyssnare för den här händelsen som går tillbaka till steg 2 när händelsen skickas. Följande kod visar de här stegen:

   ```
   flash.system.SystemUpdater.addEventListener(Event.COMPLETE, updateCompleteHandler);
   flash.system.SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
   ```

   ```
   private function updateCompleteHandler (event:Event):void {
       /*redo step 2*/
       drmContentData = new DRMContentData(metadata_bytes);
   }
   ```

   Om själva spelaren måste uppdateras skickas en status-händelse. Mer information om hur du hanterar den här händelsen finns i ”Lyssna efter en uppdateringshändelse” på sidan 545.

   Obs! *I AIR-program hanterar AIR-installationsprogrammet uppdateringen av Adobe Access-modulen och de uppdateringar av körningsversionen som behövs.*

4. Skapa avlyssnare som lyssnar efter DRMStatusEvent och DRMErrorEvent som skickas från DRMManager-objektet:

   ```
   DRMManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
   DRMManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
   ```

   Kontrollera att vouchern är giltig (inte null) i DRMStatusEvent-avlyssnaren. Hantera DRMErrorEvents i DRMErrorEvent-avlyssnaren. Se ”Använda klassen DRMStatusEvent” på sidan 537 och ”Använda klassen DRMErrorEvent” på sidan 541.

5 Läs in vouchern (licensen) som krävs för uppspelning av innehållet.

Försök först att läsa in en lokalt sparad licens för att spela upp innehållet:

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.LOCAL_ONLY)
```

När inläsningen är slutförd skickar DRMManager-objektet `DRMStatusEvent.DRM_Status`.

6 Om DRMVoucher-objektet inte är null är vouchern giltig. Gå till steg 13.

7 Om DRMVoucher-objektet är null kontrollerar du autentiseringsmetoden som krävs av innehållets policy. Använd egenskapen `DRMContentData.authenticationMethod`.

8 Om autentiseringsmetoden är `ANONYMOUS` går du vidare till steg 13.

9 Om autentiseringsmetoden är `USERNAME_AND_PASSWORD` måste programmet kunna tillhandahålla en funktion som låter användaren ange inloggningsuppgifter. Skicka dessa inloggningsuppgifter till licensservern för autentisering av användaren:

```
DRMManager.authenticate(metadata.serverURL, metadata.domain, username, password)
```

DRMManager skickar en `DRMAuthenticationErrorEvent` om autentiseringen misslyckas eller en `DRMAuthenticationCompleteEvent` om autentiseringen lyckas. Skapa avlyssnare för dessa händelser.

10 Om autentiseringsmetoden är `UNKNOWN`, måste en egen autentiseringsmetod användas. I detta fall har innehållsleverantören ordnat så att autentiseringen görs vid sidan om utan hjälp av ActionScript 3.0-API:er. Den egna autentiseringsproceduren måste skapa en autentiseringstoken som kan skickas till metoden `DRMManager.setAuthenticationToken()`.

11 Om autentiseringen misslyckas måste programmet återgå till steg 9. Kontrollera att programmet har en funktion för att hantera och begränsa upprepade autentiseringsförsök. Det kan till exempel vara ett meddelande som visas efter tre misslyckade försök och som anger att autentiseringen misslyckades och att innehållet inte kan spelas upp.

12 Om du vill använda en lagrad token istället för att fråga användaren om inloggningsuppgifter anger du en token med metoden `DRMManager.setAuthenticationToken()`. Hämta sedan licensen från licensservern och spela upp innehållet som i steg 8.

13 (Valfritt) Om autentiseringen lyckas kan du hämta in en autentiseringstoken, vilket utgörs av en cachelagrad bytearray i minnet. Hämta denna token med egenskapen `DRMAuthenticationCompleteEvent.token`. Det går att lagra och använda en autentiseringstoken så att användaren inte behöver ange sina inloggningsuppgifter flera gånger för samma innehåll. Licensservern avgör giltighetsperioden för denna autentiseringstoken.

14 Om autentiseringen lyckas hämtas licensen från servern.

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.FORCE_REFRESH)
```

När inläsningen är slutförd skickar DRMManager-objektet DRMStatusEvent.DRM_STATUS. Lyssna efter den här händelsen, när den skickas går det att spela upp innehållet.

15 Spela upp videofilmen genom att skapa ett NetStream-objekt och anropa sedan dess `play()`-metod:

```
stream = new NetStream(connection);
stream.addEventListener(DRMStatusEvent.DRM _STATUS, drmStatusHandler);
stream.addEventListener(DRMErrorEvent.DRM_ERROR, drmErrorHandler);
stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
stream.client = new CustomClient();
video.attachNetStream(stream);
stream.play(videoURL);
```

DRMContentData och sessionsobjekt

När `DRMContentData` skapas kommer det att används som ett sessionsobjekt som refererar till Flash Players DRM-modul. Alla `DRMManager`-API:er som tar emot detta `DRMContentData` kommer att använda den speciella DRM-modulen. Det finns emellertid två 2 `DRMManager`-API:er som inte använder `DRMContentData`. De är:

1. `authenticate()`
2. `setAuthenticationToken()`

Eftersom det inte finns några associerade `DRMContentData`, kommer anrop av dessa `DRMManager`-API:er att innebära att den senaste DRM-modulen från disken används. Detta kan innebära ett problem om en uppdatering av DRM-modulen inträffar under programmets DRM-arbetsflöde. Tänk dig följande scenario:

1. Programmet skapar ett `DRMContentData`-objekt `contentData1`, som använder *AdobeCP1* som en DRM-modul.
2. Programmet anropar metoden `DRMManager.authenticate(contentData1.serverURL,...)`.
3. Programmet anropar metoden `DRMManager.loadVoucher(contentData1, ...)` .

Om en uppdatering för DRM-modulen inträffar innan programmet kan komma till steg 2, kommer metoden `DRMManager.authenticate()` att avsluta autentiseringen med *AdobeCP2* som DRM-modul. Metoden `loadVoucher()` i steg 3 kommer att misslyckas eftersom *AdobeCP1* fortfarande används som DRM-modul. Uppdateringen kan ha inträffat på grund av att ett annat program anropar DRM-moduluppdateringen. Du kan undvika detta scenario genom att anropa DRM-moduluppdateringen när programmet startas.

DRM-relaterade händelser

I körningen skickas flera händelser när ett program försöker att spela upp skyddat innehåll:

- DRMDeviceGroupErrorEvent (endast AIR), skickas av DRMManager
- DRMAuthenticateEvent (endast AIR), skickas av NetStream
- DRMAuthenticationCompleteEvent, skickas av DRMManager
- DRMAuthenticationErrorEvent, skickas av DRMManager
- DRMErrorEvent, skickas av NetStream och DRMManager
- DRMStatusEvent, skickas av NetStream och DRMManager
- StatusEvent
- NetStatusEvent. Se ”Lyssna efter en uppdateringshändelse” på sidan 545

Om du vill ha stöd för innehåll skyddat av Adobe Access lägger du till händelseavlyssnare för hantering av DRM-händelserna.

Läs in vouchers i förväg för uppspelning offline

Adobe AIR 1.5 och senare

Du kan i förväg ladda vouchers (licenser) som krävs för uppspelning av innehåll som skyddas av Adobe Access. Med en förinläst voucher kan användarna visa innehållet även om de inte har en aktiv internetanslutning. (För själva förladdningen krävs dock en Internetanslutning.) Du kan använda metoden `preloadEmbeddedMetadata()` i NetStream-klassen och DRMManager-klassen för att ladda en voucher i förväg. I AIR 2.0 och senare kan du använda ett DRMContentData-objekt om du vill ladda en voucher. Den här metoden är att föredra eftersom den låter dig uppdatera DRMContentData-objektet oberoende av innehållet. (Metoden `preloadEmbeddedData()` hämtar DRMContentData från innehållet.)

Använda DRMContentData

Adobe AIR 2.0 och senare

Följande steg beskriver hur du läser in vouchern för en skyddad mediefil i förväg med ett DRMContentData-objekt.

1. Hämta binära metadata för det packade innehållet. Om du använder referenspaketeraren för Java i Adobe Access genereras denna metadatafil automatiskt med filtillägget *.metadata*. Du skulle till exempel kunna hämta dessa metadata med klassen URLLoader.
2. Skapa ett DRMContentData-objekt och skicka metadata till konstruktorfunktionen:

```
var drmData:DRMContentData = new DRMContentData( metadata );
```

3. Resten av stegen är identiska med arbetsflödet som beskrivs i ”Förstå arbetsflödet för skyddat innehåll” på sidan 529.

Använda preloadEmbeddedMetadata()

Adobe AIR 1.5 och senare

I följande steg beskrivs hur du läser in vouchern i förväg med `preloadEmbeddedMetadata()` för en DRM-skyddad mediefil:

1. Hämta och lagra mediefilen. (DRM-metadata kan bara läsas in i förväg från filer som är lagrade på lokalt.)
2. Skapa objekten NetConnection och NetStream, och ange implementeringar för callback-funktionerna `onDRMContentData()` och `onPlayStatus()` för NetStream client-objektet.
3. Skapa ett NetStreamPlayOptions-objekt och associera egenskapen `stream` med den lokala mediefilens URL.
4. Anropa `preloadEmbeddedMetadata()`-metoden i NetStream-klassen och ange NetStreamPlayOptions-objektet som identifierar den mediefil som ska tolkas.
5. Om mediefilen innehåller DRM-metadata anropas callback-funktionen `onDRMContentData()`. Metadata skickas till den här funktionen som ett DRMContentData-objekt.
6. Använd objektet DRMContentData för att erhålla vouchern med hjälp av `loadVoucher()`-metoden för DRMManager-objektet.

   Om egenskapen `authenticationMethod` för objektet DRMContentData har värdet `flash.net.drm.AuthenticationMethod.USERNAME_AND_PASSWORD` autentiserar du användaren på medierättighetsservern innan vouchern läses in. Egenskaperna `serverURL` och `domain` i DRMContentData-objektet kan överföras till `authenticate()`-metoden i DRMManager, tillsammans med användarens autentiseringsuppgifter.
7. Callback-funktionen `onPlayStatus()` anropas när filtolkningen är slutförd. Om `onDRMContentData()`-funktionen inte har anropats innehåller filen inte de metadata som krävs för att få en voucher. Det saknade anropet kan också medföra att Adobe Access inte skyddar den här filen.

Följande kodexempel för AIR illustrerar hur du läser in en voucher för en lokal mediefil i förväg:

```
package
{
import flash.display.Sprite;
import flash.events.DRMAuthenticationCompleteEvent;
import flash.events.DRMAuthenticationErrorEvent;
import flash.events.DRMErrorEvent;
import flash.ev ents.DRMStatusEvent;
import flash.events.NetStatusEvent;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.net.NetStreamPlayOptions;
import flash.net.drm.AuthenticationMethod;
import flash.net.drm.DRMContentData;
import flash.net.drm.DRMManager;
import flash.net.drm.LoadVoucherSetting;
public class DRMPreloader extends Sprite
{
    private var videoURL:String = "app-storage:/video.flv";
    private var userName:String = "user";
    private var password:String = "password";
    private var preloadConnection:NetConnection;
    private var preloadStream:NetStream;
    private var drmManager:DRMManager = DRMManager.getDRMManager();
    private var drmContentData:DRMContentData;
    public function DRMPreloader():void {
        drmManager.addEventListener(
            DRMAuthenticationCompleteEvent.AUTHENTICATION_COMPLETE,
            onAuthenticationComplete);
        drmManager.addEventListener(DRMAuthenticationErrorEvent.AUTHENTICATION_ERROR,
            onAuthenticationError);
        drmManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
        drmManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
        preloadConnection = new NetConnection();
        preloadConnection.addEventListener(NetStatusEvent.NET_STATUS, onConnect);
        preloadConnection.connect(null);
    }

    private function onConnect( event:NetStatusEvent ):void
    {
        preloadMetadata();
    }
    private function preloadMetadata():void
    {
        preloadStream = new NetStream( preloadConnection );
        preloadStream.client = this;
        var options:NetStreamPlayOptions = new NetStreamPlayOptions();
        options.streamName = videoURL;
        preloadStream.preloadEmbeddedData( options );
    }
    public function onDRMContentData( drmMetadata:DRMContentData ):void
    {
        drmContentData = drmMetadata;
        if ( drmMetadata.authenticationMethod == AuthenticationMethod.USERNAME_AND_PASSWORD )
        {
            authenticateUser();
        }
        else
```

```
            {
                getVoucher();
            }
    }
    private function getVoucher():void
    {
        drmManager.loadVoucher( drmContentData, LoadVoucherSetting.ALLOW_SERVER );
    }

    private function authenticateUser():void
    {
        drmManager.authenticate( drmContentData.serverURL, drmContentData.domain, userName,
password );
    }
    private function onAuthenticationError( event:DRMAuthenticationErrorEvent ):void
    {
        trace( "Authentication error: " + event.errorID + ", " + event.subErrorID );
    }

    private function onAuthenticationComplete( event:DRMAuthenticationCompleteEvent ):void
    {
        trace( "Authenticated to: " + event.serverURL + ", domain: " + event.domain );
        getVoucher();
    }
    private function onDRMStatus( event:DRMStatusEvent ):void
    {
        trace( "DRM Status: " + event.detail);
        trace("--Voucher allows offline playback = " + event.isAvailableOffline );
        trace("--Voucher already cached          = " + event.isLocal );
        trace("--Voucher required authentication = " + !event.isAnonymous );
    }
    private function onDRMError( event:DRMErrorEvent ):void
    {
        trace( "DRM error event: " + event.errorID + ", " + event.subErrorID + ", " + event.text );
    }
    public function onPlayStatus( info:Object ):void
    {
        preloadStream.close();
    }
}
}
```

DRM-relaterade medlemmar och händelser i NetStream-klassen

Flash Player 10.1 och senare, Adobe AIR 1.0 och senare

Klassen NetStream tillhandahåller en enkelriktad direktuppspelningsanslutning mellan Flash Player eller ett AIR-program, och antingen Flash Media Server eller det lokala filsystemet. (Klassen NetStream har också stöd för progressiv nedladdning.) Ett NetStream-objekt är en kanal i ett NetConnection-objekt. Klassen NetStream skickar fyra DRM-relaterade händelser:

Händelse	Beskrivning
drmAuthenticate (endast AIR)	Definieras i klassen DRMAuthenticateEvent. Den här händelsen skickas när ett NetStream-objekt försöker att spela upp skyddat innehåll som kräver inloggningsuppgifter för autentisering före uppspelning. Egenskaperna för den här händelsen omfattar header, usernamePrompt, passwordPrompt och urlPrompt som kan användas för att hämta och ställa in inloggningsuppgifter. Den här händelsen inträffar upprepade gånger tills NetStream-objektet tar emot giltiga inloggningsuppgifter.
drmError	Definieras i klassen DRMErrorEvent och skickas när ett NetStream-objekt försöker att spela upp skyddat innehåll och ett DRM-relaterat fel uppstår. Exempelvis skickas ett DRM-felhändelseobjekt när användarautentiseringen misslyckas. Det här felet kan inträffa om användaren inte har köpt rätten att visa innehållet. Det kan också inträffa om innehållets utgivare inte har stöd för visningsprogrammet.
drmStatus	Definieras i klassen DRMStatusEvent. Den här händelsen skickas när det skyddade innehållet börjar spelas upp (när användaren är autentiserad och har rätt att spela upp innehållet). DRMStatusEvent-objektet innehåller information relaterat till vouchern. Det är till exempel information om huruvida innehållet kan göras tillgängligt offline eller när vouchern utgår och innehållet inte längre kan visas.
status	Definieras i events.StatusEvent och skickas endast när programmet försöker att spela upp skyddat innehåll genom att anropa metoden NetStream.play(). Värdet för statuskodsegenskapen är ”DRM.encryptedFLV”.

Klassen NetStream innehåller följande DRM-specifika metoder, som endast kan användas i AIR:

Metod	Beskrivning
resetDRMVouchers()	Tar bort alla lokalt cachelagrade DRM-voucherdata (Digital Rights Management). Programmet måste hämta relevanta vouchers igen för att användaren ska kunna komma åt det krypterade innehållet. Till exempel tar följande kod bort alla voucher från cachen: NetStream.resetDRMVouchers();
setDRMAuthenticationCredentials()	Skickar en uppsättning inloggningsuppgifter, det vill säga användarnamn, lösenord och autentiseringstyp, till NetStream-objektet för autentisering. Giltiga autentiseringstyper är `"drm"` och `"proxy"`. Med autentiseringstypen `"drm"` autentiseras de angivna inloggningsuppgifterna mot Adobe Access. Med autentiseringstypen `"proxy"` autentiseras inloggningsuppgifterna mot proxyservern och de måste stämma överens med inloggningsuppgifterna som krävs av proxyservern. Ett företag kan till exempel kräva att programmet autentiserar mot en proxyserver innan användaren kan börja använda Internet. Proxyalternativet tillåter den här typen av autentisering. Om inte anonym autentisering används måste användaren fortfarande autentisera mot Adobe Access efter proxyautentiseringen för att få vouchern och spela upp innehållet. Du kan använda `setDRMAuthenticationCredentials()` en andra gång med `"drm"`-alternativet för att autentisera mot Adobe Access.
preloadEmbeddedMetadata()	Tolkar en lokal mediefil med inbäddade metadata. Om DRM-relaterade metadata påträffas anropar AIR callback-funktionen `onDRMContentData()`.

I AIR anropar dessutom ett NetStream-objekt återanropsfunktionerna `onDRMContentData()` och `onPlayStatus()` som ett resultat av ett anrop till metoden `preloadEmbeddedMetaData()`. Funktionen `onDRMContentData()` anropas när DRM-metadata hittas i en mediefil. Funktionen `onPlayStatus()` anropas när filen har tolkats. Funktionerna `onDRMContentData()` och `onPlayStatus()` måste definieras för `client`-objektet som associeras med NetStream-instansen. Om samma NetStream-objekt används för att ladda vouchern i förväg och för uppspelning av innehållet väntar du på `onPlayStatus()`-anropet som genereras av `preloadEmbeddedMetaData()` innan du påbörjar uppspelningen.

I följande kod för AIR ställs användarnamn (”administrator”), lösenord (”password”) och autentiseringstypen ”drm” in för autentisering av användaren. Metoden setDRMAuthenticationCredentials() måste tillhandahålla inloggningsuppgifter som stämmer överens med de av innehållsleverantören kända och accepterade inloggningsuppgifterna. Dessa uppgifter är samma inloggningsuppgifter som gör det möjligt för användaren att se innehållet. Koden för att spela upp videofilmen och säkerställa anslutningen till videoströmmen tas inte med här.

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Använda klassen DRMStatusEvent

Flash Player 10.1, Adobe AIR 1.0 och senare

Ett NetStream-objekt skickar ett DRMStatusEvent-objekt när innehållet som skyddas av Adobe Access kan börja spelas upp. (Om innehållet går att spela upp indikerar det att licensen har verifierats och att användaren är autentiserad och har rätt att visa innehållet). DRMStatusEvent skickas också för anonyma användare om de ges åtkomst. Licensen kontrolleras för att verifiera ifall anonyma användare som inte behöver autentiseras ska ha åtkomst för uppspelning av innehållet. Anonyma användare kan nekas åtkomst av flera anledningar. En anonym användare har till exempel inte tillgång till innehållet efter att licensen har upphört.

DRMStatusEvent-objektet innehåller information relaterat till licensen. Det är till exempel information om huruvida licensen kan göras tillgängligt offline eller när vouchern utgår och innehållet inte längre kan visas. Programmet kan använda dessa data för att förmedla användarens policystatus och behörighet.

DRMStatusEvent-egenskaper

Flash Player 10.1, Adobe AIR 1.0 och senare

Klassen DRMStatusEvent innehåller följande egenskaper. Vissa egenskaper har blivit tillgängliga i senare versioner av AIR än 1.0. För fullständig versionsinformation, se *Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen*.

För egenskaper som inte stöds i Flash Player 10.1 tillhandahåller klassen DRMVoucher liknande egenskaper för Flash Player.

Egenskap	Beskrivning
contentData	Ett DRMContentData-objekt med DRM-metadata inbäddat i innehållet.
detail (endast AIR)	En sträng förklarar innehållet i statushändelsen. I DRM 1.0 är DRM.voucherObtained det enda giltiga värdet.
isAnonymous (endast AIR)	Anger om innehållet som skyddas av Adobe Access ska vara tillgängligt utan att användaren måste ange sina inloggningsuppgifter (true) eller inte (false). Värdet false innebär att en användare måste ange ett användarnamn och ett lösenord som stämmer överens med de av innehållsleverantören kända och förväntade uppgifterna.
isAvailableOffline (endast AIR)	Anger om innehållet som skyddas med Adobe Access kan göras tillgängligt offline (true) eller inte (false). Om du vill att digitalt skyddat innehåll ska finnas tillgängligt offline måste verifikationen cachas till användarens lokala dator.
isLocal	Anger om den voucher som krävs för uppspelning av innehållet är cachelagrad lokalt.

Egenskap	Beskrivning
offlineLeasePeriod (endast AIR)	Antalet återstående dagar som innehållet kan visas offline.
policies (endast AIR)	Ett anpassat objekt som kan innehålla anpassade DRM-egenskaper.
voucher	DRMVoucher.
voucherEndDate (endast AIR)	Datumet då verifikationen förfaller och innehållet inte längre kan visas.

Skapa en DRMStatusEvent-hanterare

Flash Player 10.1, Adobe AIR 1.0 och senare

I följande exempel skapas en händelsehanterare som förmedlar DRM-innehållets statusinformation till NetStream-objektet som händelsen härstammar från. Lägg till den här händelsehanteraren i ett NetStream-objekt som pekar på skyddat innehåll.

```
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
```

Använda klassen DRMAuthenticateEvent

Adobe AIR 1.0 och senare

DRMAuthenticateEvent-objektet skickas när ett NetStream-objekt försöker att spela upp skyddat innehåll som kräver inloggningsuppgifter för autentisering före uppspelning.

DRMAuthenticateEvent-hanteraren samlar in nödvändiga inloggningsuppgifter (som användarnamn, lösenord och typ) och skickar värdena till `NetStream.setDRMAuthenticationCredentials()`-metoden för verifiering. Varje AIR-program måste ha en funktion som erhåller inloggningsuppgifter. Programmet kan till exempel ha ett enkelt användargränssnitt där användaren kan ange sitt användarnamn och lösenord. Ha också med en funktion för hantering och begränsning av upprepade autentiseringsförsök.

DRMAuthenticateEvent-egenskaper

Adobe AIR 1.0 och senare

Klassen DRMAuthenticateEvent innehåller följande egenskaper:

Egenskap	Beskrivning
authenticationType	Anger om inloggningsuppgifterna gäller för autentisering mot Adobe Access (drm) eller en proxyserver (proxy). Proxyalternativet innebär till exempel att programmet autentiserar mot en proxyserver om det krävs innan användaren kan börja använda Internet. Om inte anonym autentisering används måste användaren fortfarande autentisera mot Adobe Access efter proxyautentiseringen för att få vouchern och spela upp innehållet. Du kan använda setDRMAuthenticationcredentials() en andra gång med "drm"-alternativet för att autentisera mot Adobe Access.
huvud	Filhuvudet med krypterat innehåll som tillhandahålls av servern. Det innehåller information om det krypterade innehållet. Denna rubriksträng kan skickas vidare till ett Flash-program för att få programmet att skapa en dialogruta för användarnamn och lösenord. Rubriksträngen kan användas som instruktion i dialogrutan. Rubriken kan till exempel vara "Ange ditt användarnamn och lösenord".
netstream	NetStream-objektet som initierade den här händelsen.
passwordPrompt	En uppmaning om ett lösenord, tillhandahålls av servern. Strängen kan innehålla instruktioner för det lösenord som krävs.
urlUppmaning	Ett kommando för en URL tillhandahålls av servern. Strängen kan ange platsen dit användarnamnet och lösenordet ska skickas.
usernamePrompt	En uppmaning för ett användarnamn; tillhandahålls av servern. Strängen kan innehålla instruktioner för det användarnamn som krävs. Innehållsleverantören kan till exempel kräva en e-postadress som användarnamn.

De tidigare nämnda strängarna tillhandahålls endast av FMRMS-servern. I Adobe Access Server används inte dessa strängar.

Skapa en DRMAuthenticateEvent-hanterare

Adobe AIR 1.0 och senare

I följande exempel skapas en händelsehanterare som skickar en uppsättning hårdkodade inloggningsuppgifter till NetStream-objektet som utlöste händelsen. (Koden för att spela upp videofilmen och säkerställa anslutningen till videoströmmen tas inte med här.)

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Skapa ett gränssnitt för att hämta inloggningsuppgifter

Adobe AIR 1.0 och senare

Om det skyddade innehållet kräver användarautentisering måste AIR-programmet vanligtvis hämta användarens inloggningsuppgifter via ett användargränssnitt.

Nedan finns ett Flex-exempel på ett enkelt gränssnitt som hämtar inloggningsuppgifter. Det består av ett panelobjekt som innehåller två TextInput-objekt, ett för användarnamnet och ett för lösenordet. Panelen innehåller också en knapp som startar metoden `credentials()`.

```
<mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute" title="Login">
    <mx:TextInput x="110" y="46" id="uName"/>
    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
    <mx:Text x="35" y="48" text="Username:"/>
    <mx:Text x="35" y="78" text="Password:"/>
    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
</mx:Panel>
```

Metoden `credentials()` är en användardefinierad metod som skickar värdena för användarnamnet och lösenordet till metoden `setDRMAuthenticationCredentials()`. När värdena har skickats återställer metoden `credentials()` värdena på TextInput-objekten.

```
<mx:Script>
    <![CDATA[
        public function credentials():void
        {
            videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
            uName.text = "";
            pWord.text = "";
        }
    ]]>
</mx:Script>
```

Ett sätt att implementera den här typen av enkelt gränssnitt är att inkludera panelen i ett nytt läge. Det nya läget bygger på grundläget när DRMAuthenticateEvent-objektet skapas. Följande exempel innehåller ett VideoDisplay-objekt med ett källattribut som pekar på en skyddad FLV-fil. I det här fallet ändras metoden `credentials()` så att den även returnerar programmet till grundläget. Metoden gör det efter att användarens inloggningsuppgifter har skickats och värdena för TextInput-objektet har återställts.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="800"
    height="500"
    title="DRM FLV Player"
    creationComplete="initApp()" >

    <mx:states>
        <mx:State name="LOGIN">
            <mx:AddChild position="lastChild">
                    <mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute"
                            title="Login">
                    <mx:TextInput x="110" y="46" id="uName"/>
                    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
                    <mx:Text x="35" y="48" text="Username:"/>
                    <mx:Text x="35" y="78" text="Password:"/>
                    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
                    <mx:Button x="193" y="115" label="Reset" click="uName.text='';
                            pWord.text='';"/>
                </mx:Panel>
            </mx:AddChild>
        </mx:State>
    </mx:states>
```

```
    <mx:Script>
        <![CDATA[
                import flash.events.DRMAuthenticateEvent;
            private function initApp():void
            {
                videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                                        drmAuthenticateEventHandler);
            }

            public function credentials():void
            {
                videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
                uName.text = "";
                pWord.text = "";
                currentState='';
            }

            private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
            {
                currentState='LOGIN';
            }
        ]]>
    </mx:Script>

    <mx:VideoDisplay id="video" x="50" y="25" width="700" height="350"
        autoPlay="true"
        bufferTime="10.0"
        source="http://www.example.com/flv/Video.flv" />
</mx:WindowedApplication>
```

Använda klassen DRMErrorEvent

Flash Player 10.1 och senare, Adobe AIR 1.0 och senare

Adobe Flash Player och Adobe AIR skickar ett DRMErrorEvent-objekt när ett NetStream-objekt försöker att spela upp skyddat innehåll och ett DRM-relaterat fel uppstår. Om inloggningsuppgifterna är ogiltiga i ett AIR-program skickas DRMAuthenticateEvent-objektet upprepade gånger tills användaren anger giltiga inloggningsuppgifter eller tills fler försök nekas i programmet. Programmet ska lyssna på andra DRM-felhändelser för att hitta, identifiera och hantera DRM-relaterade fel.

Även om användaren har giltiga inloggningsuppgifter kan villkoren i innehållets voucher ändå hindra användaren från att visa det krypterade innehållet. En användare kan till exempel nekas åtkomst för att ha försökt visa innehåll i ett oauktoriserat program. Ett oauktoriserat program är ett program som utgivaren av det krypterade innehållet inte har verifierat. I så fall skickas ett DRMErrorEvent-objekt.

Felhändelserna kan också utlösas om innehållet är skadat eller om programmets version inte motsvarar det som anges i vouchern. Programmet måste ha lämplig mekanism för att hantera fel.

DRMErrorEvent-egenskaper

Flash Player 10.1 och senare, Adobe AIR 1.0 och senare

En fullständig lista över fel finns i Felkoder vid körning i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen. De DRM-relaterade felen börjar vid nummer 3300.

Skapa en DRMErrorEvent-hanterare

Flash Player 10.1 och senare, Adobe AIR 1.0 och senare

I följande exempel skapas en händelsehanterare för NetStream-objektet som utlöste händelsen. Den anropas när NetStream påträffar ett fel vid ett försök att spela upp det skyddade innehållet. När ett program påträffar ett fel utförs vanligtvis ett antal rensningsåtgärder. Sedan informeras användaren om felet, och denne kan välja mellan olika alternativ för att lösa problemet.

```
private function drmErrorEventHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
```

Använda klassen DRMManager

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

Använd klassen DRMManager för att hantera vouchers och sessioner på mediarättighetsservern i program.

Voucher-hantering (endast AIR)

När en användare spelar upp skyddat innehåll erhålls och cachelagras den licens som krävs för visning av innehållet i körningen. Om filen sparas lokalt, och licensen tillåter offlineuppspelning, kan användaren visa innehållet i AIR-programmet. Sådan lokal offlineuppspelning lyckas även om det inte finns någon anslutning till medieservern. Genom att använda DRMManager och NetStream-metoden `preloadEmbeddedMetadata()` kan du cachelagra vouchern i förväg. Programmet behöver inte få licensen som krävs för att visa innehållet. Programmet kan exempelvis hämta mediefilen och sedan erhålla vouchern när användaren fortfarande är online.

Om du vill förhandsladda en voucher hämtar du ett DRMContentData-objekt med hjälp av `preloadEmbeddedMetadata()`-metoden för NetStream-objektet. DRMContentData-objektet innehåller URL-adressen och domänen för den mediarättighetsserver som kan ge ut licensen och anger om användarautentisering krävs. Med hjälp av den här informationen kan du anropa `loadVoucher()`-metoden för DRMManager-objektet för att hämta och cachelagra vouchern. Anvisningar om hur du läser in vouchers i förväg beskrivs i detalj i avsnittet ”Läs in vouchers i förväg för uppspelning offline” på sidan 532.

Sessionshantering

Du kan också autentisera användaren mot en mediarättighetsserver och hantera beständiga sessioner med hjälp av DRMManager.

Upprätta en session med mediarättighetsservern genom att anropa `authenticate()`-metoden för DRMManager-objektet. När autentiseringen är färdig skickar DRMManager ett DRMAuthenticationCompleteEvent-objekt. Det här objektet innehåller en sessionstoken. Du kan spara denna token för att upprätta framtida sessioner så att användaren slipper uppge sina kontouppgifter. Skicka token till metoden `setAuthenticationToken()` när du vill upprätta en ny autentiserad session. (Inställningarna på den server som genererade token avgör när den upphör att gälla och dess övriga attribut. AIR-programkod bör inte tolka tokens datastruktur, eftersom strukturen kan ändras i framtida AIR-uppdateringar.)

En autentiseringstoken kan överföras till andra datorer. Du kan skydda tokens genom att lagra dem på den krypterade lokala lagringsplatsen i AIR. Du hittar mer information i avsnittet ”Krypterad lokal lagringsplats” på sidan 712.

DRMStatus-händelser

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

DRMManager skickar ett DRMStatusEvent-objekt när ett anrop till `loadVoucher()`-metoden har slutförts.

Om en voucher erhålls kommer egenskapen `detail` (endast AIR) i händelseobjektet att ha värdet: "DRM.voucherObtained", och egenskapen `voucher` innehålla DRMVoucher-objektet.

Om en voucher inte erhålls har egenskapen `detail` (endast AIR) fortfarande värdet "DRM.voucherObtained", men egenskapen `voucher` är `null`. En voucher kan inte erhållas om du exempelvis använder LoadVoucherSetting för *localOnly* och det inte finns någon lokalt cachelagrad voucher.

Om `loadVoucher()`-anropet inte slutförs korrekt, kanske på grund av ett autentiserings- eller kommunikationsfel, skickar DRMManager i stället ett DRMErrorEvent-objekt eller ett DRMAuthenticationErrorEvent-objekt.

DRMAuthenticationComplete-händelser

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

DRMManager skickar ett DRMAuthenticationCompleteEvent-objekt om en användare autentiseras genom ett anrop till `authenticate()`-metoden.

DRMAuthenticationCompleteEvent-objektet innehåller en token som kan återanvändas för användarautentiseringen i olika programsessioner. Överför denna token till `setAuthenticationToken()`-metoden i DRMManager för att återupprätta sessionen. (Den som skapar en token anger även dess attribut, till exempel förfallodatum. Adobe tillhandahåller inte något API för granskning av tokenattribut.)

DRMAuthenticationError-händelser

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

DRMManager skickar ett DRMAuthenticationErrorEvent-objekt om en användare inte kan autentiseras via ett anrop till `authenticate()`- eller `setAuthenticationToken()`-metoden.

Använda DRMContentData-klassen

Flash Player 10.1 och senare, Adobe AIR 1.5 och senare

DRMContentData-objektet innehåller metadata-egenskaperna för innehåll som skyddas av Adobe Access. DRMContentData-egenskaperna innehåller den information som behövs för att erhålla en licensvoucher som krävs för att visa innehållet. Du kan använda klassen DRMContentData för att hämta metadata-filen som är associerad med innehållet, enligt anvisningarna i ”Utförligt API-arbetsflöde” på sidan 530.

Du hittar mer information i avsnittet om klassen DRMContentData i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen

Uppdatera Flash Player för stöd av Adobe Access

Flash Player 10.1 och senare

Viktigt: I Flash Player 11.5 och senare integreras Adobe Access-modulen så uppdateringssteget (anrop av `SystemUpdater.update(SystemUpdaterType.DRM)`) är inte nödvändigt. Detta gäller följande webbläsare och plattformar:

- ActiveX-kontroll för Flash Player 11.5, för alla plattformar utom Internet Explorer i Windows 8
- Plugin-program för Flash Player 11.5, för alla webbläsare
- Adobe AIR (dator och mobil)

Detta innebär att uppdateringssteget *fortfarande krävs* i följande fall:

- Internet Explorer i Windows 8
- Flash Player 11.4 och senare, utom Google Chrome 22 och senare (alla plattformar) eller 21 och senare (Windows)

Obs! *Det är fortfarande säkert att anropa* `SystemUpdater.update(SystemUpdaterType.DRM)` *i ett system med Flash Player 11.5 eller senare, men inget laddas ned.*

För stöd av Adobe Access krävs en Adobe Access-modul i Flash Player. När Flash Player försöker att spela upp skyddat innehåll indikerar körningen om modulen eller en ny version av Flash Player måste hämtas. På detta sätt ger Flash Player SWF-utvecklare alternativet att inte uppdatera om det inte är önskvärt.

För att spela upp skyddat innehåll vill SWF-utvecklare för det mesta uppdatera till den Adobe Access-modul eller spelare som krävs. För att uppdatera till den senaste versionen av Adobe Access-modulen eller Flash Player kan du använda API:et SystemUpdater.

API:et SystemUpdater tillåter endast en uppdatering åt gången. Felkoden 2202 indikerar att en uppdatering redan pågår i den aktuella körningsinstansen eller en annan instans. Om en uppdatering till exempel pågår i en Flash Player-instans i Internet Explorer går det inte att uppdatera i en Flash Player-instans som körs i Firefox.

API:et SystemUpdater stöds endast för skrivbordsplattformar.

Obs! *För tidigare versioner av Flash Player än 10.1 använder du uppdateringsfunktionen som stöds för de tidigare spelarversionerna (manuell nerladdning och installation från www.adobe.com eller ExpressInstall). Nödvändiga uppdateringar för Adobe Access hanteras i AIR-installationsprogrammet och API:et SystemUpdater stöds inte.*

Lyssna efter en uppdateringshändelse

Flash Player 10.1 och senare

När en uppdatering av Adobe Access-modulen krävs skickar NetStream-objektet en NetStatusEvent-händelse med kodvärdet `DRM.UpdateNeeded`. Detta värde indikerar att NetStream-objektet inte kan spela upp den skyddade strömmen med några av de Adobe Access-moduler som finns installerade. Lyssna efter den här händelsen och anropa följande kod:

```
SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

Den här koden uppdaterar Adobe Access-modulen som finns installerad i spelaren. Användaren behöver inte godkänna uppdateringen av den här modulen.

Om Adobe Access-modulen inte hittas genereras ett fel. Se steg 3 i ”Utförligt API-arbetsflöde” på sidan 530.

Obs! *Om play() anropas på en krypterad ström i tidigare spelare än 10.1 skickas en NetStatusEvent med kodvärdet NetStream.Play.StreamNotFound. För tidigare spelare använder du uppdateringsfunktionen som stöds för den spelaren (manuell nerladdning och installation från www.adobe.com eller ExpressInstall).*

När en uppdatering av själva spelaren krävs skickar SystemUpdater-objektet en StatusEvent med kodvärdet `DRM.UpdateNeededButIncompatible`. Användarens godkännande krävs för att uppdatera spelaren. Tillhandahåll ett gränssnitt i programmet där användaren kan godkänna och påbörja uppdateringen av spelaren. Lyssna efter händelsen StatusEvent och anropa följande kod:

```
SystemUpdater.update(flash.system.SystemUpdaterType.SYSTEM);
```

Den här koden initierar uppdateringen av spelaren.

Ytterligare händelser för klassen SystemUpdater beskrivs i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

När uppdateringen av spelaren har slutförts omdirigeras användaren till sidan där uppdateringen började. Adobe Access-modulen hämtas och strömmen kan börja spelas upp.

Externa licenser

Flash Player 11 och senare, Adobe AIR 3.0 och senare

Genom att lagra vouchern (licensen) på disken och i minnet med hjälp av metoden `storeVoucher` kan licenser även erhållas utan att en Adobe Access-licensserver behöver användas.

För att kunna spela upp krypterade videofilmer i Flash Player och AIR, måste DRM-vouchern för den videon erhållas för respektive miljö. DRM-vouchern innehåller videons krypteringsnyckel och den skapas i den Adobe Access-licensserver som kunden distribuerat.

Flash Player-/AIR-miljön erhåller vanligtvis denna voucher genom att en voucher-förfrågan skickas till Adobe Access-licensservern som anges i videons DRM-metadata (klassen `DRMContentData`). Flash-/AIR-programmet kan utlösa denna licensförfrågan genom att metoden `DRMManager.loadVoucher()` anropas. Om det inte finns någon licens för innehållet på disken eller i minnet, kommer en licensbegäran automatiskt från Flash Player-/AIR-miljön när uppspelningen av den krypterade videon startar. I båda fallen påverkas Flash-/AIR-programmets prestanda av kommunikationen med Adobe Access-licensservern.

`DRMManager.storeVoucher()` tillåter att Flash-/AIR-programmet skickar DRM-vouchers som erhållits vid sidan om till Flash Player-/AIR-miljön. Processen med en licensförfrågan kan sedan hoppas över i körningsmiljön och vidareskickade vouchers för spelning av krypterade videofilmer kan användas. DRM-vouchern måste fortfarande skapas i Adobe Access-licensservern innan den kan hämtas på annat sätt. Du kan emellertid välja att lägga en voucher på valfri HTTP-server, i stället för på en mer offentlig Adobe Access-licensserver.

`DRMManager.storeVoucher()` används även för att ge stöd åt DRM-voucher-delning mellan fler enheter. I Adobe Access 3.0 kallas denna funktion för "domänstöd". Om du i din distribution har stöd för detta användningssätt, kan du registrera flera datorer för enhetsgruppen med hjälp av metoden `DRMManager.addToDeviceGroup()`. Om det finns en dator med en giltig domänbunden voucher för ett givet innehåll, kan AIR-programmet extrahera serialiserade DRM-vouchers med metoden `DRMVoucher.toByteArray()`, och metoden `DRMManager.storeVoucher()` kan sedan användas på andra datorer för att importera vouchers.

Enhetsregistrering

En DRM-voucher är bunden till slutanvändarens dator. Därav följer att Flash-/AIR-programmen måste ha ett unikt ID för användarens dator för att kunna referera till rätt serialiserade DRM-voucher-objekt. I följande scenario beskrivs en enhetsregistreringsprocess:

Anta att du har utfört följande åtgärder:

- Du har ställt in SDK:n för Adobe Access-servern.
- Du har ställt in en HTTP-server för att erhålla förgenererade licenser.
- Du har skapat ett Flash-program för att visa det skyddade innehållet.

I enhetsregistreringsfasen ingår följande åtgärder:

1. Flash-programmet skapar ett slumpmässigt ID.
2. Flash-programmet anropar metoden `DRMManager.authenticate()`. Programmet måste innehålla det slumpmässigt genererade ID:t i en autentiseringsförfrågan. Inkludera till exempel ID:t i användarnamnsfältet.
3. Åtgärden som omnämns i steg 2 resulterar i att Adobe Access skickar en autentiseringsförfrågan till kundens server. Denna förfrågan innehåller enhetscertifikatet.
   a. Servern extraherar enhetscertifikatet och det genererade ID:t från förfrågan och sparar det.
   b. Kundens undersystem förgenererar licenser för detta enhetscertifikat, sparar dem och beviljar åtkomst för dem på ett sätt som associerar dem med det genererade ID:t.
4. Från servern kommer ett meddelande om att förfrågan lyckades.
5. Flash-programmet sparar det genererade ID:t lokalt i ett LSO-objekt (Local Shared Object).

Efter enhetsregistreringen, använder Flash-programmet det genererade ID:t på samma sätt som enhets-ID:t skulle ha använts i det föregående schemat:

1. Flash-programmet försöker hitta det genererade ID:t i LSO-objektet.
2. Om det genererade ID:t hittas, kommer Flash-programmet att använda detta ID för att hämta de förgenererade licenserna. Flash-programmet skickar licenserna till Adobe Access-klienten för att användas med metoden `DRMManager.storeVoucher()`.
3. Om det genererade ID:t inte hittas, kommer Flash-programmet att gå igenom enhetsregistreringsproceduren.

Återställning till standardinställningar

När enhetsanvändaren anropar alternativet för återställning till standardinställningar, kommer enhetscertifikatet att rensas. För att du ska kunna spela skyddat innehåll måste Flash-programmet gå igenom enhetsregistreringsproceduren ytterligare en gång. Om Flash-programmet skickar en inaktuell förgenererad licens, kommer Adobe Access-klienten att avvisa den eftersom licensen krypterades för ett äldre enhets-ID.

Domänstöd

Flash Player 11 och senare, Adobe AIR 3.0 och senare

Om innehållets metadata anger att en domänregistrering krävs, kan AIR-programmet anropa en API för att den ska ingå i en enhetsgrupp. Denna åtgärd utlöser en domänregistreringsförfrågan som skickas till domänservern. När en licens har utfärdats till en enhetsgrupp, kan den exporteras och delas med andra enheter som ingår i enhetsgruppen.

Informationen för enhetsgruppen används sedan i objektet `VoucherAccessInfo` för `DRMContentData`. Detta används därefter för att presentera den information som krävs för att hämta och använda en voucher.

Spela upp krypterat innehåll med domänstöd

Gör så här för att spela upp krypterat innehåll med Adobe Access:

1. Använd `VoucherAccessInfo.deviceGroup` och kontrollera om en enhetsgruppsregistrering krävs.
2. Om autentisering krävs:
   - **a** Använd egenskapen `DeviceGroupInfo.authenticationMethod` för att ta reda på om en autentisering krävs.
   - **b** Om du måste utföra en autentisering, autentiserar du användaren genom att utföra ETT av följande steg:
     - Hämta användarens användarnamn och lösenord. Anropa `DRMManager.authenticate(deviceGroup.serverURL, deviceGroup.domain, username, password)`.
     - Hämta en cache-lagrad/förgenererad autentiseringstoken och anropa `DRMManager.setAuthenticationToken()`.
   - **c** Anropa `DRMManager.addToDeviceGroup()`.
3. Hämta vouchern för innehållet genom att utföra en av följande åtgärder:
   - **a** Använd metoden `DRMManager.loadVoucher()`.
   - **b** Hämta vouchern från en annan enhet registrerad i samma enhetsgrupp. Tillhandahåll vouchern för `DRMManager` genom metoden `DRMManager.storeVoucher()`.
4. Spela upp krypterat innehåll med metoden `NetStream.play()`.

Om du vill exportera licensen för innehållet, kan alla enheter ge dig licensens rå-byte med metoden `DRMVoucher.toByteArray()` sedan licensen erhållits från Adobe Access-licensservern. Innehållsleverantörer begränsar vanligtvis antalet enheter i en enhetsgrupp. Om gränsvärdet uppnås, kanske du måste anropa en ej använd enhet med metoden `DRMManager.removeFromDeviceGroup()` innan den aktuella enheten kan registreras.

Förhandsgranska licensen

Flash-programmet kan skicka en licensförhandsgranskningsförfrågan, vilket innebär att programmet kan genomföra en förhandsgranskning innan användaren tillfrågas om att köpa innehållet. Detta görs för att bestämma om användarens dator uppfyller de krav som ställs för att uppspelningen ska fungera. Licensförhandsgranskning avser klientens möjlighet att förhandsgranska licensen (för att se de behörigheter som gäller för licensen) till skillnad mot att förhandsvisa innehållet (visa en liten del av innehållet innan beslut om köp fattas). Vissa av de parametrar som är unika för varje dator är: utdatamöjligheter och skyddsstatus, tillgänglig körnings-/DRM-version och DRM-klientens säkerhetsnivå. Licensens förhandsgranskningsläge gör det möjligt för körnings-/DRM-klienten att testa licensserverns affärslogik och ge information tillbaka till användaren så att han/hon kan fatta ett välgrundat beslut. Klienten kan således upptäcka hur en giltig licens ser ut, men erhåller inte den faktiska nyckeln för att dekryptera innehållet. Stöd för licensförhandsgranskning är valfritt och endast nödvändigt om du implementerar ett eget program som använder denna funktionalitet.

Leverera innehåll

Adobe Access känner inte av leveransförfarandet för innehållet eftersom Flash Player gör utdrag ur nätverkslagret och helt enkelt tillhandahåller det skyddade innehållet till undersystemet för Adobe Access. Innehållet kan således levereras med HTTP, dynamisk HTTP-direktuppspelning, RTMP eller RTMPE.

Du kan emellertid råka ut för vissa problem på grund av behovet av det skyddade innehållets metadata (vanligtvis i form av en ".metadata"-fil) innan Adobe Access kan erhålla en licens för att dekryptera innehållet. Detta gäller speciellt för FLV- och F4V-data, med protokollen RTMP/RTMPE, som endast kan levereras till klienten genom FMS-servern (Flash Media Server). På grund av detta måste klienten hämta metadata-blobben på andra sätt. Ett sätt att lösa detta problem är att ha metadata på en HTTP-webbserver och implementera klientens videospelare så att den hämtar lämpliga metadata beroende på det innehåll som spelas upp.

```
private function getMetadata():void{

    extrapolated-path-to-metadata = "http://metadatas.mywebserver.com/" + videoname;
    var urlRequest : URLRequest = new URLRequest(extrapolated-path-to-the-metadata + ".metadata");
    var urlStream : URLStream = new URLStream();
    urlStream.addEventListener(Event.COMPLETE, handleMetadata);
    urlStream.addEventListener(IOErrorEvent.NETWORK_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.IO_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.VERIFY_ERROR, handleIOError);
    try{
        urlStream.load(urlRequest);
     }catch(se:SecurityError){
        videoLog.text += se.toString() + "\n";
     }catch(e:Error){
        videoLog.text += e.toString() + "\n";
     }
}
```

Open Source Media Framework

OSMF (Open Source Media Framework) är ett ActionScript-baserat ramverk som ger dig fullständig flexibilitet och kontroll när det gäller att skapa egna fantasirika medieupplevelser. Mer information om OSMF finns på webbplatsen för OSMF-utvecklare.

Arbetsflöde för uppspelning av skyddat innehåll

1 Skapa en `MediaPlayer`-instans.

```
player = new MediaPlayer();
```

2 Registrera händelsen `MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE` för spelaren. Denna händelse kommer att skickas om innehållet är DRM-skyddat.

```
player.addEventListener(MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE,
onDRMCapabilityChange);
```

3 Erhåll `DRMTrait`-instansen i händelsehanteraren. `DRMTrait` är gränssnittet genom vilket du anropar DRM-relaterade metoder, till exempel `authenticate()`. När du läser in ett DRM-skyddat innehåll, utför OSMF DRM-valideringen och skickar lägeshändelser. Lägg till en `DRMEvent.DRM_STATE_CHANGE`-händelsehanterare för att hantera `DRMTrait`.

```
private function onDRMCapabilityChange
    (event :MediaPlayerCapabilityChangeEvent) :void
        {
            if (event.type == MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE
                && event.enabled)
            {
                drmTrait = player.media.getTrait(MediaTraitType.DRM) as DRMTrait;
                drmTrait.addEventListener
                    (DRMEvent.DRM_STATE_CHANGE, onDRMStateChange);
            }
        }
```

4 Hantera DRM-händelserna i metoden `onDRMStateChange()`.

```
        private function onDRMStateChange(event :DRMEvent) :void
        {
            trace ( "DRMState: ",event.drmState);
            switch(event.drmState)
            {
                case DRMState.AUTHENTICATION_NEEDED:
                    // Identity-based content
                    var authPopup :AuthWindow = AuthWindow.create(_parentWin);
                    authPopup.serverURL = event.serverURL;
                    authPopup.addEventListener("dismiss", function () :void {
                        trace ("Authentication dismissed");
                        if(_drmTrait != null)
                        {
                            //Ignore authentication. Just
                            //try to acquire a license.
                            _drmTrait.authenticate(null, null);
                        }
                    });
                    authPopup.addEventListener("authenticate",
                                    function (event :AuthWindowEvent) :void {
                        if(_drmTrait != null)
```

```
                {
                    _drmTrait.authenticate(event.username, event.password);
                }
            });
            authPopup.show();
            break;
        case DRMState.AUTHENTICATING:
            //Display any authentication message.
            trace("Authenticating...");
            break;
        case DRMState.AUTHENTICATION_COMPLETE:
            // Start to retrieve voucher and playback.
            // You can display the voucher information at this point.
            if(event.token)
            // You just received the authentication token.
            {
                trace("Authentication success. Token: \n", event.token);
            }
            else
            // You have got the voucher.
            {
                trace("DRM License:");
                trace("Playback window period: ",
                    !isNaN(event.period) ? event.period == 0 ?
                    "<unlimited>" : event.period : "<none>");
                trace("Playback window end date: ",
                    event.endDate != null ? event.endDate : "<none>");
                trace("Playback window start date: ",
                    event.startDate != null ? event.startDate : "<none>");
            }
            break;
        case DRMState.AUTHENTICATION_ERROR:
            trace ("DRM Error:", event.mediaError.errorID +
                "[" + DRMErrorEventRef.getDRMErrorMnemonic
                (event.mediaError.errorID) + "]");
            //Stop everything.
            player.media = null;
            break;
        case DRMState.DRM_SYSTEM_UPDATING:
            Logger.log("Downloading DRM module...");
            break;
        case DRMState.UNINITIALIZED:
            break;
    }
}
```

Kapitel 28: Lägga till PDF-innehåll i AIR

Adobe AIR 1.0 och senare

Program som körs i Adobe® AIR® kan visa både SWF-, HTML- och PDF-innehåll. AIR-program återger PDF-innehåll med hjälp av klassen HTMLLoader, motorn WebKit och webbläsarplug-inen Adobe® Reader®. I ett AIR-program kan PDF-innehåll sträckas ut över hela programmets höjd och bredd eller som en del av gränssnittet. Visningen av PDF-filer i ett AIR-program styrs av Adobe Reader-pluginprogrammet för webbläsare. Ändringar i gränssnittet för Reader-verktygsfältet (till exempel kontroller för placering, ankarpunkter och synlighet) gäller även i efterföljande visning av PDF-filer i både AIR-program och i webbläsaren.

Viktigt! *Användaren måste ha Adobe Reader eller Adobe® Acrobat® version 8.1 eller en senare version installerad för att kunna återge PDF-innehåll i AIR.*

Identifiera PDF-funktion

Adobe AIR 1.0 och senare

Om användaren inte har Adobe Reader eller Adobe Acrobat 8.1 eller senare kan inte PDF-innehåll visas i ett AIR-program. Om du vill se om en användare kan återge PDF-innehåll ska du först kontrollera egenskapen `HTMLLoader.pdfCapability`. Den här egenskapen är inställd på någon av följande konstanter i klassen HTMLPDFCapability:

Konstant	Beskrivning
HTMLPDFCapability.STATUS_OK	En version (8.1 eller senare) av Adobe Reader som stöds har hittats och PDF-innehåll kan läsas in i ett HTMLLoader-objekt.
HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND	Ingen version av Adobe Reader har hittats. Det går inte att visa PDF-innehåll i ett HTMLLoader-objekt.
HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD	Adobe Reader har upptäckts, men versionen är för gammal. Det går inte att visa PDF-innehåll i ett HTMLLoader-objekt.
HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD	En version (8.1 eller senare) av Adobe Reader som stöds har hittats, men den version av Adobe Reader som har konfigurerats för att hantera PDF-innehåll är äldre än Reader 8.1. Det går inte att visa PDF-innehåll i ett HTMLLoader-objekt.

Om användaren har Windows och kör Adobe Acrobat eller Adobe Reader version 7.x eller senare, används den versionen även om en senare version med stöd för inläsning av PDF-innehåll installeras. Om värdet för egenskapen `pdfCapability` är `HTMLPDFCapability.STATUS_OK` när ett AIR-program gör ett försök att läsa in PDF-innehåll, visar den äldre versionen av Acrobat eller Reader en varning (och inget undantag genereras i AIR-programmet). Om det här är en situation som kan uppstå för slutanvändarna bör du kanske ge dem anvisningar om att stänga Acrobat under tiden som de kör programmet. Du kan visa de här anvisningarna om PDF-innehållet inte går att läsa in inom en viss tid.

I Linux söker AIR efter Adobe Reader på den sökväg (PATH) som exporterats av användaren (om den innehåller kommandot acroread) och i katalogen /opt/Adobe/Reader.

Följande kod identifierar om en användare kan visa PDF-innehåll i ett AIR-program. Om användaren inte kan visa PDF-innehåll spåras felkoden som motsvarar felobjektet för HTMLPDFCapability:

```
 if(HTMLLoader.pdfCapability == HTMLPDFCapability.STATUS_OK)
{
    trace("PDF content can be displayed");
}
else
{
    trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

Läsa in PDF-innehåll

Adobe AIR 1.0 och senare

Du kan lägga till en PDF i ett AIR-program genom att skapa en HTMLLoader-instans, ange dess mått och läsa in sökvägen till en PDF.

Följande exempel läser in en PDF från en extern plats. Ersätt URLRequest med sökvägen till en tillgänglig, extern PDF.

```
 var request:URLRequest = new URLRequest("http://www.example.com/test.pdf");
pdf = new HTMLLoader();
pdf.height = 800;
pdf.width = 600;
pdf.load(request);
container.addChild(pdf);
```

Du kan också läsa in innehåll från filens URL:er och AIR-specifika URL-scheman, t.ex. app och app-storage. Följande kod läser till exempel in filen test.pdf i PDF:ens underkatalog i programkatalogen:

app:/js_api_reference.pdf

Läs mer om URL-scheman i AIR i avsnittet ”URI-scheman” på sidan 815.

Skriptning av PDF-innehåll

Adobe AIR 1.0 och senare

Du kan använda JavaScript för att kontrollera PDF-innehåll på samma sätt som på en sida i en webbläsare.

JavaScript-tillägg i Acrobat ger bland annat följande funktioner:

- Kontroll av sidnavigering och förstoring
- Bearbetning av formulär i dokumentet
- Kontroll av multimediehändelser

Mer information om JavaScript-tillägg för Adobe Acrobat finns i Adobe Acrobat Developer Connection på http://www.adobe.com/devnet/acrobat/javascript.html.

Grundläggande om kommunikation mellan HTML och PDF

Adobe AIR 1.0 och senare

JavaScript på en HTML-sida kan skicka ett meddelande till JavaScript i PDF-innehåll genom att anropa metoden `postMessage()` i det DOM-objekt som representerar PDF-innehållet. Ta till exempel följande inbäddade PDF-innehåll:

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

Följande JavaScript-kod som finns i HTML-innehållet skickar ett meddelande till JavaScript-koden i PDF-filen:

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

PDF-filen kan inkludera JavaScript för att ta emot det här meddelandet. Du kan lägga till JavaScript-kod i PDF-filer i vissa kontexter, inklusive dokument-, mapp-, sid-, fält- och gruppnivåkontexterna. Det är bara kontexten på dokumentnivå som definierar de skript som utvärderas när PDF-dokumentet öppnas som tas upp här.

En PDF-fil kan lägga till en `messageHandler`-egenskap till `hostContainer`-objektet. Egenskapen `messageHandler` är ett objekt som definierar hanterarfunktioner som svarar på meddelanden. Följande kod definierar till exempel funktionen som hanterar meddelanden som tas emot av PDF-filen från värdbehållaren (som är det HTML-innehåll som är inbäddat i PDF-filen):

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

JavaScript-koden på HTML-sidan kan anropa metoden `postMessage()` för det PDF-objekt som finns på sidan. När den här metoden anropas skickas ett meddelande (`"Hello from HTML"`) till JavaScript på dokumentnivå i PDF-filen:

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

Ett mer avancerat exempel och för information om hur du använder Acrobat 8 för att lägga till JavaScript i en PDF-fil finns i Korsskriptning av PDF-innehåll i Adobe AIR.

Skriptning av PDF-innehåll från ActionScript

Adobe AIR 1.0 och senare

ActionScript-kod (i SWF-innehåll) kan inte kommunicera direkt med JavaScript i PDF-innehåll. ActionScript kan dock kommunicera med JavaScript på den HTML-sida som är inläst i ett HTMLLoader-objekt som läser in PDF-innehåll. Den JavaScript-koden kan också kommunicera med JavaScript i den inlästa PDF-filen. Mer information finns i avsnittet ”Programmera HTML och JavaScript i AIR” på sidan 985.

Kända begränsningar för PDF-innehåll i AIR

Adobe AIR 1.0 och senare

PDF-innehåll i Adobe AIR har följande begränsningar:

- PDF-innehåll visas inte i ett fönster (ett NativeWindow-objekt) som är genomskinligt (om egenskapen `transparent` är angiven som `true`).
- Visningsordningen för en PDF-fil fungerar annorlunda mot andra visningsobjekt i ett AIR-program. Även om PDF-innehåll hålls ihop korrekt enligt HTML-visningsordningen så kommer det alltid att visas över innehåll i AIR-programmets visningsordning.

- Om vissa visuella egenskaper i ett HTMLLoader-objekt som innehåller ett PDF-dokument ändras, visas inte PDF-dokumentet. Exempel på dessa egenskaper är egenskaperna `filters`, `alpha`, `rotation` och `scaling`. Om dessa egenskaper ändras visas inte PDF-filen förrän egenskaperna har återställts. PDF-innehållet visas inte heller om du ändrar de här egenskaperna för de behållare för visningsobjekt som innehåller HTMLLoader-objektet.
- PDF-innehåll visas bara om egenskapen `scaleMode` för det av Stage-objektets NativeWindow-objekt som innehåller PDF-innehållet har värdet `StageScaleMode.NO_SCALE`. Om egenskapen har ett annat värde visas inte PDF-innehållet.
- Om du klickar på länkar till innehåll i PDF-filen uppdateras rullningspositionen för PDF-innehållet. Om du klickar på länkar till innehåll utanför PDF-filen dirigeras det HTMLLoader-objekt som innehåller PDF:en om (även om länkens mål är ett nytt fönster).
- PDF-kommentarer kan inte användas i AIR.

Kapitel 29: Grunderna i användarinteraktion

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ditt program kan skapa interaktivitet via ActionScript 3.0 för att svara på användaråtgärder. Observera att du för det här avsnittet förutsätts ha grundläggande kunskaper om ActionScript 3.0-händelsemodellen. Mer information finns i ”Hantera händelser” på sidan 125.

Samla in indata från användarna

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Användarinteraktion, oberoende av om det är via tangentbord, mus, kamera eller en kombination av alla tre, utgör själva grunden för all interaktivitet. I ActionScript 3.0 innebär att identifiera och besvara användarinteraktion huvudsakligen att avlyssna händelser.

Klassen InteractiveObject, en underklass till klassen DisplayObject, ger en grundläggande struktur för händelser och funktioner som är nödvändiga för att hantera användarinteraktionen. Du skapar inte direkt en instans av klassen InteractiveObject. I stället ärver visningsobjekt som SimpleButton, Sprite, TextField och olika Flash-utvecklingsverktyg och Flex-komponenter sina respektive användarinteraktionsmodeller från den här klassen och de delar därför en gemensam struktur. Detta betyder att de tekniker som du lär dig och de koder som du skriver för att hantera användarinteraktionen i ett objekt, härlett från InteractiveObject, även går att använda för de övriga.

Viktiga termer och begrepp

Det är viktigt att du känner till följande terminologi för användarinteraktion innan du fortsätter:

Teckenkod En numerisk kod som representerar ett tecken i den aktuella teckenuppsättningen (associerad till en tangent som trycks ned på tangentbordet). ”D” och ”d” är exempelvis olika teckenkoder trots att de skapas med samma tangent på ett standardtangentbord.

Snabbmeny Den meny som öppnas när användaren högerklickar eller använder en speciell tangentbords- och muskombination. Kommandona på snabbmenyn gäller vanligtvis för det som användaren klickar på. En snabbmeny för en bild kan exempelvis innehålla kommandon för att visa bilden i ett separat fönster och för att ladda ned den.

Fokus Indikationen på att det markerade elementet är aktivt och att det är målet för tangentbords- eller musinteraktionen.

Tangentkod En numerisk kod som motsvarar en fysisk tangent på tangentbordet.

Fler hjälpavsnitt

InteractiveObject

Keyboard

Mouse

ContextMenu

Hantera fokus

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett interaktivt objekt kan erhålla fokus antingen via programkoden eller genom en användarinteraktion. Om dessutom egenskapen `tabEnabled` är `true`, kan användaren flytta fokus från ett objekt till ett annat med tabbtangenten. Observera att värdet `tabEnabled` som standard är `false` förutom i följande fall:

- För ett SimpleButton-objekt är värdet `true` .
- För ett textinmatningsfält är värdet `true`.
- För ett Sprite- eller ett MovieClip-objekt, med `buttonMode` satt till `true`, är värdet `true`.

I alla dessa situationer kan du lägga till en avlyssnare för `FocusEvent.FOCUS_IN` eller `FocusEvent.FOCUS_OUT` för att erhålla ytterligare beteenden när fokus ändras. Detta är speciellt användbart för textfält och formulär, men det kan också användas för spriter, filmklipp eller objekt som ärver från klassen InteractiveObject. I exemplet nedan visas hur du aktiverar fokusändringar med tabbtangenten och hur du svarar på den efterföljande fokushändelsen. I detta fall ändras färgen på varje ruta när den erhåller fokus.

***Obs!** Eftersom kortkommandon används för att hantera fokus i Flash Professional bör du testa SWF-filer i en webbläsare eller AIR, och inte i Flash, för att få korrekt fokussimulering.*

```
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
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Identifiera indatatyper

Flash Player 10.1 och senare, Adobe AIR 2 och senare

I Flash Player 10.1 och Adobe AIR 2 introducerades en möjlighet att testa körningsmiljön för stöd av särskilda indatatyper. Du kan använda ActionScript om du vill testa ifall enheten på vilken körningen för närvarande finns:

- Stöder indata med styluspenna eller finger (eller ingen pekdata alls).
- Har ett virtuellt eller fysiskt tangentbord för användaren (eller inget tangentbord alls).
- Visar en markör (om ingen markör visas fungerar inte funktioner som bygger på att en markör hålls över ett objekt).

ActionScript API:er för identifiering av indatatyper omfattar:

- Egenskapen flash.system.Capabilities.touchScreenType: Ett värde som anges vid körning och indikerar vilken indatatyp som stöds i den aktuella miljön.
- Klassen flash.system.TouchscreenType: En klass med uppräkningsvärdekonstanter för egenskapen Capabilities.touchscreenType.
- Egenskapen flash.ui.Mouse.supportsCursor: Ett värde ges vid körningen som indikerar om en ständig markör är tillgänglig eller inte.
- Egenskapen flash.ui.Keyboard.physicalKeyboardType: Ett värde ges vid körningen som indikerar om ett fullständigt fysiskt tangentbord finns tillgängligt eller endast en numerisk knappsats, eller inget tangentbord alls.
- Klassen flash.ui.KeyboardType: En klass av numreringsvärdekonstanter för egenskapen flash.ui.Keyboard.physicalKeyboardType.
- Egenskapen flash.ui.Keyboard.hasVirtualKeyboard: Ett värde ges vid körningen som indikerar om ett virtuellt tangentbord finns tillgängligt för användaren (antingen istället för ett fysiskt tangentbord eller utöver det fysiska tangentbordet).

Genom API:erna för identifiering av indata kan du dra nytta av användarens enhetsfunktioner, eller tillhandahålla alternativ om användaren saknar de aktuella funktionerna. Dessa API:er är särskilt användbara vid utveckling av mobila program och program med pekfunktioner. Om du till exempel använder ett gränssnitt för en mobil enhet med små knappar som är avsedda för en styluspenna kan du tillhandahålla ett alternativt gränssnitt med större knappar för användare som använder pekfunktioner med fingret. Följande kod är för ett program med funktionen createStylusUI(), som tilldelar en uppsättning användargränssnittselement som passar för styluspennor. En annan funktion, kallad createTouchUI(), tilldelar en annan uppsättning användargränssnittselement som passar för pekfunktioner som utförs med fingrarna:

```
if(Capabilities.touchscreenType == TouchscreenType.STYLUS ){
    //Construct the user interface using small buttons for a stylus
    //and allow more screen space for other visual content
    createStylusUI();
} else if(Capabilities.touchscreenType = TouchscreenType.FINGER){
    //Construct the user interface using larger buttons
    //to capture a larger point of contact with the device
    createTouchUI();
}
```

Tänk på följande kompatibilitetstabell när du utvecklar program för miljöer med olika indatatyper:

Miljö	supportsCursor	touchscreenType == FINGER	touchscreenType == STYLUS	touchscreenType == NONE
Vanlig dator	true	false	false	true
Enheter med kapacitiv pekskärm (tablet-datorer, handdatorer och telefoner som känner av mänsklig beröring, till exempel Apple iPhone eller Palm Pre)	false	true	false	false
Enheter med resistiv pekskärm (tablet-datorer, handdatorer och telefoner som känner av exakt punktberöring, till exempel HTC Fuze)	false	false	true	false
Enheter utan pekskärm (telefoner och enheter som kan köra program men som inte har någon skärm som kan känna av beröring)	false	false	false	true

Obs! *Olika enhetsplattformar kan ha stöd för flera olika kombinationer av indatatyper. Se den här tabellen som en allmän vägledning.*

Kapitel 30: Tangentbordsindata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ditt program kan hämta och svara på tangentbordsindata och kan hantera en IME så att användare kan skriva icke-ASCII-text på multibytespråk. Observera att du för det här avsnittet förutsätts ha grundläggande kunskaper om ActionScript 3.0-händelsemodellen. Mer information finns i ”Hantera händelser” på sidan 125.

Se ”Identifiera indatatyper” på sidan 558 för mer information om hur du under körning identifierar vilket typ av tangentbordsstöd som finns tillgängligt (fysiskt, virtuellt, alfanumeriskt, eller numeriskt med 12 knappar o.s.v.).

Ett redigeringsprogram för inmatningsmetod (IME-program) gör att användarna kan skriva komplexa tecken och symboler med hjälp av ett standardtangentbord. Du kan använda IME-klasserna för att göra det möjligt för användarna att utnyttja fördelarna med sina system-IME i dina program.

Fler hjälpavsnitt

flash.events.KeyboardEvent

flash.system.IME

Samla in tangentbordsindata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Visningsobjekt som ärver interaktionsmodellen från klassen InteractiveObject kan använda händelseavlyssnare för att svara på tangentbordshändelser. Du kan exempelvis placera en händelseavlyssnare på scenen för att lyssna och svara på tangentbordsinmatningar. I följande exempel samlar en händelseavlyssnare in en tangentnedtryckning och sedan visas tangentnamnet och egenskaperna för tangentkoden:

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Vissa tangenter, till exempel Ctrl-tangenten, genererar händelser trots att den inte har någon glyfrepresentation.

I det föregående exemplet samlar tangentbordsavlyssnaren in tangentbordsinmatningar för hela scenen. Du kan också skriva en händelseavlyssnare för ett specifikt visningsobjekt på scenen. Den här händelseavlyssnaren aktiveras när objektet är i fokus.

I följande exempel återspeglas tangentnedtryckningar i utdatapanelen bara när användaren skriver något i TextField-instansen. När du håller ned skifttangenten ändras textfältets ramfärg temporärt till rött.

För koden görs antagandet att det redan finns en TextField-instans med namnet `tf` på scenen.

```
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;
    }
}
```

Klassen TextField rapporterar också en `textInput`-händelse som du kan avlyssna när användaren skriver text. Mer information finns i ”Hämta textinmatningar” på sidan 378.

Obs! *I AIR-miljön kan en tangentbordshändelse avbrytas. I Flash Player-miljön kan en tangentbordshändelse inte avbrytas.*

Tangentkoder och teckenkoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med egenskaperna `keyCode` och `charCode` kan du ta reda på vilken tangent som trycks ned och därefter aktivera olika åtgärder. Egenskapen `keyCode` är ett numeriskt värde som motsvarar värdet på tangenten på tangentbordet. Egenskapen `charCode` är ett numeriskt värde för tangenten i den aktuella teckenuppsättningen. (Standardteckenuppsättningen är UTF-8, som har stöd för ASCII.)

Den största skillnaden mellan tangentkoder och teckenvärden är att ett tangentkodsvärde representerar en viss tangent på tangentbordet (tangenten ”1” på ett numeriskt tangentbord är inte densamma som ”1” på den översta raden på tangentbordet, men själva tangenten som genererar ”1” och tangenten som genererar ”!” är densamma) och att teckenvärdet representerar ett speciellt tecken (tecken ”R” och ”r” är olika).

Obs! *Mer information om mappningen mellan tangenter och deras teckenkodvärden i ASCII finns i avsnittet för klassen flash.ui.Keyboard i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.*

Mappningen mellan tangenter och deras tangentkoder beror på den aktuella enheten och operativsystemet. Du ska därför inte använda tangentmappning för att aktivera åtgärder. Du ska i stället använda de fördefinierade konstanterna i klassen Keyboard för att referera till rätt `keyCode`-egenskap. I stället för att exempelvis använda tangentmappning för skifttangenten ska du använda konstanten `Keyboard.SHIFT` (vilket framgår av exemplet nedan).

Prioritet för tangentbordshändelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Precis som för andra händelser bestäms tangentbordshändelsen av visningsobjektshierarkin och inte av i vilken ordning metoderna för `addEventListener()` är tilldelade i koden.

I nästa exempel placerar du ett textfält med namnet `tf` i ett filmklipp med namnet `container` och lägger till en händelseavlyssnare för en tangentbordshändelse i båda instanserna:

```
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 + ")");
}
```

Eftersom det finns en avlyssnare i båda textfälten och i dess överordnade behållare, anropas funktionen `reportKeyDown()` två gånger för varje tangentnedtryckning i TextField. Observera, att för varje tangentnedtryckning skickas en händelse från textfältet innan filmklippet `container` skickar en händelse.

Tangentbordshändelsen bearbetas i operativsystemet och webbläsaren innan den bearbetas i Adobe Flash Player eller AIR. Om du i exempelvis Microsoft Internet Explorer trycker på Ctrl+W stängs webbläsarfönstret innan en eventuell SWF-fil skickar en tangentbordshändelse.

Använda klassen IME

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen IME gör att du kan hantera operativsystemets IME-program i Flash Player eller Adobe AIR.

Med ActionScript kan du kontrollera följande:

- Om ett IME-program är installerat på användarens dator (`Capabilities.hasIME`)
- Om IME-programmet är aktiverat eller inaktiverat på användarens dator (`IME.enabled`)
- Vilket konverteringsläge som används av aktuellt IME-program (`IME.conversionMode`)

Du kan koppla ett indatatextfält till en bestämd IME-kontext. När du växlar mellan indatafält kan du även växla IME-programmet mellan Hiragana (japanska), fullbredds numeriskt, halvbredds numeriskt, direktinmatning osv.

Med ett IME kan användare skriva tecken som saknar ASCII-kodning, till exempel kinesiska, japanska och koreanska.

Mer information om att arbeta med IME-program finns i dokumentationen för det operativsystem som du utvecklar programmet för. Fler resurser finns på följande webbplatser:

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/library/mac/navigation/
- http://www.java.sun.com/

Obs! *Om IME-programmet inte är aktivt på användarens dator går det inte att anropa andra IME-metoder eller IME-egenskaper än* `Capabilities.hasIME`*. När du har aktiverat IME-programmet manuellt, fungerar efterföljande ActionScript-anrop till IME-metoder och IME-egenskaper. Om du till exempel använder ett japansk IME-program måste du aktivera det innan du kan anropa någon IME-metod eller IME-egenskap.*

Kontrollera om ett IME-program är installerat och aktiverat

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Innan du anropar någon IME-metod eller IME-egenskap bör du alltid kontrollera om det finns ett IME-program installerat och aktiverat på användarens dator. Följande kod visar hur du kontrollerar om användaren har ett IME-program installerat och aktiverat innan du anropar någon metod:

```
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.");
}
```

Föregående kod kontrollerar först om användaren har ett IME-program installerat med hjälp av egenskapen `Capabilities.hasIME`. Om egenskapen är inställd på `true` kommer koden sedan att kontrollera om användarens IME-program är aktiverat. Detta görs med hjälp av egenskapen `IME.enabled`.

Kontrollera vilket IME-konverteringsläge som är aktiverat

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar flerspråkiga program kan du behöva kontrollera vilket konverteringsläge som är aktivt hos användaren. Följande kod visar hur du kontrollerar om användaren har ett IME-program installerat och, om så är fallet, vilket IME-konverteringsläge som är aktivt:

```
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.";
}
```

Föregående kod kontrollerar först om användaren har ett IME-program installerat. Sedan kontrollerar koden vilket konverteringsläge det aktuella IME-programmet använder genom att jämföra egenskapen `IME.conversionMode` med alla konstanter i klassen IMEConversionMode.

Ställa in IME-konverteringsläget

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du ändrar konverteringsläget för användarens IME-program måste du se till att koden placeras inom ett `try..catch`-block, eftersom en inställning av konverteringsläget med hjälp av egenskapen `conversionMode` kan ge ett fel om IME-programmet inte kan ställa in konverteringsläget. I följande kod visas hur du använder ett `try..catch`-block när du ställer in egenskapen `IME.conversionMode`:

```
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;
    }
}
```

Föregående kod skapar först ett textfält, som används för att visa ett statusmeddelande för användaren. Om IME-programmet är installerat aktiverar koden sedan IME-programmet och ställer in konverteringsläget på koreanska. Om användarens datorn inte har ett koreanskt IME-program installerat, genereras ett fel av Flash Player eller AIR som fångas upp av `try..catch`-block. Blocket `try..catch` visar felmeddelandet i det tidigare skapade textfältet.

Inaktivera IME-programmet för vissa textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I vissa fall kanske du vill inaktivera användarens IME-program medan de skriver tecken. Om du till exempel har ett textfält som bara godtar numerisk inmatning, kanske du inte vill att IME-programmet ska visas och göra att inmatningen av data tar lång tid.

Följande exempel visar hur du kan lyssna efter händelserna `FocusEvent.FOCUS_IN` och `FocusEvent.FOCUS_OUT` och inaktivera användarens IME-program på lämpligt sätt:

```
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;
    }
}
```

Det här exemplet skapar två indatatextfält, `phoneTxt` och `nameTxt`, och lägger sedan till två händelseavlyssnare till textfältet `phoneTxt`. När användaren ställer in fokus på textfältet `phoneTxt` skickas en `FocusEvent.FOCUS_IN`-händelse och sedan inaktiveras IME-programmet. När textfältet `phoneTxt` förlorar fokus, skickas händelsen `FocusEvent.FOCUS_OUT` för att aktivera IME-programmet igen.

Lyssna efter IME-kompositionshändelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

IME-kompositionshändelser skickas när en kompositionssträng ställs in. Om användaren till exempel har sitt IME-program aktiverat och aktivt och skriver en sträng på japanska, skickas händelsen `IMEEvent.IME_COMPOSITION` så fort användaren väljer kompositionssträngen. För att kunna lyssna efter händelsen `IMEEvent.IME_COMPOSITION` måste du lägga till en händelseavlyssnare till den statiska händelsen `ime` i klassen System (`flash.system.System.ime.addEventListener(...)`), som i följande exempel:

```
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;
}
```

I föregående kod skapas två textfält som läggs till i visningslistan. Det första textfältet, `inputTxt`, är ett indatatextfält som gör att användaren kan skriva japansk text. Det andra fältet, `outputTxt`, är ett dynamiskt textfält som visar felmeddelanden för användaren eller visar de japanska tecken som användaren skriver i textfältet `inputTxt`.

Virtuella tangentbord

Flash Player 10.2 och senare, AIR 2.6 och senare

Mobilenheter, som telefoner och bärbara enheter, har ofta ett virtuellt, programvarubaserat tangentbord i stället för ett fysiskt. Med klasserna i API:t för Flash kan du:

- Identifiera när ett virtuellt tangentbord öppnas och stängs.
- Hindra tangentbordet från att öppnas.
- Identifiera det område på scenen som det virtuella tangentbordet skymmer.

- Skapa interaktiva objekt som öppnar tangentbordet när de får fokus. (Stöds inte i AIR-program på iOS.)
- (Endast AIR) Inaktivera automatisk panorering så att programmet kan ändra visningen beroende på tangentbordet.

Kontrollera beteende för virtuellt tangentbord

Det virtuella tangentbordet öppnas automatiskt när användaren pekar i ett textfält eller ett särskilt konfigurerat interaktivt objekt. När tangentbordet öppnas följs de inbyggda plattsformsreglerna för panorering och storleksändring av programinnehållet så att användare ser texten när de skriver.

När tangentbordet öppnas skickar det fokuserade objektet följande händelser i sekvens:

Händelsen `softKeyboardActivating` – skickas direkt innan tangentbordet börjar höjas över scenen. Om du anropar metoden `preventDefault()` för det skickade händelseobjektet öppnas inte det virtuella tangentbordet.

Händelsen `softKeyboardActivate` – skickas när `softKeyboardActivating`-händelsehantering har slutförts. När det fokuserade objektet skickar den här händelsen har egenskapen `softKeyboardRect` för Stage-objektet uppdaterats för att återspegla det område på scenen som skyms av det virtuella tangentbordet. Den här händelsen kan inte avbrytas.

Obs! *Om tangentbordet exempelvis ändrar storlek när användaren ändrar tangentbordstypen skickar det fokuserade objektet en andra softKeyboardActivate-händelse.*

Händelsen `softKeyboardDeactivate` – skickas när det virtuella tangentbordet stängs av någon anledning. Den här händelsen kan inte avbrytas.

Följande exempel lägger till två TextField-objekt på scenen. Det övre TextField-objektet förhindrar att tangentbordet höjs när du pekar på fältet och stänger det om det redan är höjt. Det nedre TextField-objektet demonstrerar standardbeteendet. I exemplet rapporteras de skärmtangentbordshändelser som skickas av båda textfälten.

```
package
{
import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.SoftKeyboardEvent;
public class SoftKeyboardEventExample extends Sprite
{
    private var tf1:TextField = new TextField();
    private var tf2:TextField = new TextField();

    public function SoftKeyboardEventExample()
    {
        tf1.width = this.stage.stageWidth;
        tf1.type = TextFieldType.INPUT;
        tf1.border = true;
        this.addChild( tf1 );

        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, preventSoftKe
yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, preventSoftKe
yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, preventSoftKeyboard
);

        tf2.border = true;
        tf2.type = TextFieldType.INPUT;
```

```
        tf2.width = this.stage.stageWidth;
        tf2.y = tf1.y + tf1.height + 30;
        this.addChild( tf2 );

        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, allowSoftKeyboard);
    }

    private function preventSoftKeyboard( event:SoftKeyboardEvent ):void
    {
            event.preventDefault();
            this.stage.focus = null; //close the keyboard, if raised
            trace( "tf1 dispatched: " + event.type + " -- " + event.triggerType );
    }
    private function allowSoftKeyboard( event:SoftKeyboardEvent )    :void
    {
            trace( "tf2 dispatched: " + event.type + " -- " + event.triggerType );
    }
}
}
```

Lägga till virtuellt tangentbordsstöd för interaktiva objekt

Flash Player 10.2 and och senare, AIR 2.6 och senare (men stöds inte på iOS)

Normalt öppnas det virtuella tangentbordet bara när någon pekar på ett TextField-objekt. Du kan konfigurera en instans av klassen InteractiveObject för att öppna det virtuella tangentbordet när det får fokus.

Om du vill konfigurera en InteractiveObject-instans så att den öppnar det virtuella tangentbordet anger du dess `needsSoftKeyboard`-egenskap som `true`. Så snart objektet tilldelas scenens focus-egenskap öppnas det virtuella tangentbordet automatiskt. Du kan dessutom öppna tangentbordet genom att anropa metoden `requestSoftKeyboard()` för InteractiveObject.

Följande exempel visar hur du kan programmera ett InteractiveObject-objekt så att det fungerar som ett fält för textinmatning. Den TextInput-klass som visas i exemplet anger egenskapen `needsSoftKeyboard` så att tangentbordet öppnas när det behövs. Objektet lyssnar efter `keyDown`-händelser och infogar det tecken som skrivs i fältet.

I exemplet används textmotorn i Flash för att lägga till och visa text som skrivs och för att hantera vissa viktiga händelser. För enkelhets skull implementeras inte alla funktioner för textfältet i exemplet.

```
package  {
    import flash.geom.Rectangle;
    import flash.display.Sprite;
    import flash.text.engine.TextElement;
    import flash.text.engine.TextBlock;
    import flash.events.MouseEvent;
    import flash.events.FocusEvent;
    import flash.events.KeyboardEvent;
    import flash.text.engine.TextLine;
    import flash.text.engine.ElementFormat;
    import flash.events.Event;

    public class TextInput extends Sprite
    {

        public var text:String = " ";
        public  var textSize:Number = 24;
        public var textColor:uint = 0x000000;
        private var _bounds:Rectangle = new Rectangle( 0, 0, 100, textSize );
        private var textElement: TextElement;
        private var textBlock:TextBlock = new  TextBlock();

        public function TextInput( text:String = "" )
        {
            this.text = text;
            this.scrollRect = _bounds;
            this.focusRect= false;

            //Enable keyboard support
            this.needsSoftKeyboard = true;
            this.addEventListener(MouseEvent.MOUSE_DOWN, onSelect);
            this.addEventListener(FocusEvent.FOCUS_IN, onFocusIn);
            this.addEventListener(FocusEvent.FOCUS_OUT, onFocusOut);

            //Setup text engine
            textElement = new TextElement( text, new ElementFormat( null, textSize, textColor ) );
            textBlock.content = textElement;
            var firstLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8 );
            firstLine.x = 4;
            firstLine.y = 4 + firstLine.totalHeight;
            this.addChild( firstLine );

        }

        private function onSelect( event:MouseEvent ):void
        {
            stage.focus = this;
        }
        private function onFocusIn( event:FocusEvent ):void
        {
            this.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onFocusOut( event:FocusEvent ):void
        {
            this.removeEventListener( KeyboardEvent.KEY_UP, onKey );
        }
```

```
        private function onKey( event:KeyboardEvent ):void
        {
            textElement.replaceText( textElement.text.length, textElement.text.length,
String.fromCharCode( event.charCode ) );
            updateText();
        }
        public function set bounds( newBounds:Rectangle ):void
        {
            _bounds = newBounds.clone();
            drawBackground();
            updateText();
            this.scrollRect = _bounds;

            //force update to focus rect, if needed
            if( this.stage!= null && this.focusRect && this.stage.focus == this )
                this.stage.focus = this;
        }

        private function updateText():void
        {
            //clear text lines
            while( this.numChildren > 0 ) this.removeChildAt( 0 );

            //and recreate them
            var textLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8);
            while ( textLine)
            {
                textLine.x = 4;
                if( textLine.previousLine != null )
                {
                    textLine.y = textLine.previousLine.y +
                                 textLine.previousLine.totalHeight + 2;
                }
                              else
                {
                    textLine.y = 4 + textLine.totalHeight;
                }
                this.addChild(textLine);
                textLine = textBlock.createTextLine(textLine, _bounds.width - 8 );
            }
        }

        private function drawBackground():void
        {
            //draw background and border for the field
            this.graphics.clear();
            this.graphics.beginFill( 0xededed );
            this.graphics.lineStyle( 1, 0x000000 );
            this.graphics.drawRect( _bounds.x + 2, _bounds.y + 2, _bounds.width - 4,
_bounds.height - 4);
            this.graphics.endFill();
        }
    }
}
```

Följande huvudprogramklass visar hur du använder klassen TextInput och hanterar programlayouten när tangentbordet öppnas eller enhetens orientering ändras. Huvudklassen skapar ett TextInput-objekt och anger dess gränser så att det fyller scenen. Klassen justerar storleken på TextInput-objektet antingen när det virtuella tangentbordet öppnas eller när scenens storlek ändras. Klassen lyssnar efter tangentbordshändelser från TextInput-objektet och storleksändringshändelser från scenen. Oavsett orsaken till händelsen fastställer programmet scenens synliga område och ändrar inmatningskontrollen för att fylla det. I ett riktigt program behöver du naturligtvis en mer sofistikerad layoutalgoritm.

```
package  {

    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.geom.Rectangle;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;

    public class CustomTextField extends MovieClip {

        private var customField:TextInput = new TextInput("Input text: ");

        public function CustomTextField() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;
            this.addChild( customField );
            customField.bounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );

            //track soft keyboard and stage resize events
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onDisplayAreaChange );
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onDisplayAreaChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            //Fill the stage if possible, but avoid the area covered by a keyboard
            var desiredBounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );
            if( this.stage.stageHeight - this.stage.softKeyboardRect.height <
desiredBounds.height )
                desiredBounds.height = this.stage.stageHeight -
this.stage.softKeyboardRect.height;

            customField.bounds = desiredBounds;
        }
    }
}
```

Obs! *Scenen skickar bara storleksändringshändelser som svar på en orienteringsändring när egenskapen* `scaleMode` *är inställd på* `noScale`*. I andra lägen ändras inte scenens dimensioner, utan i stället skalas innehållet för att passa.*

Hantera programvisningsändringar

AIR 2.6 och senare

I AIR kan du inaktivera det standardbeteende för panorering och storleksändring som är kopplat till att ett virtuellt tangentbord öppnas genom att ställa in `softKeyboardBehavior`-elementet i programbeskrivningen på `none`:

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

När du inaktiverar det automatiska beteendet måste ditt program se till att göra nödvändiga ändringar av programvisningen. En softKeyboardActivate-händelse skickas när tangentbordet öppnas. När `softKeyboardActivate`-händelsen skickas innehåller egenskapen `softKeyboardRect` för scenen dimensionerna för det område som döljs av det öppna tangentbordet. Använd de här dimensionerna för att flytta eller ändra storlek på innehållet, så att det visas korrekt när tangentbordet är öppet och användaren skriver text. (När tangentbordet är stängt är alla softKeyboardRect-rektangelns dimensioner noll.)

När tangentbordet stängs skickas en `softKeyboardDeactivate`-händelse, och du kan återställa programvisningen till den normala.

```
package  {
    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;
    import flash.display.InteractiveObject;
    import flash.text.TextFieldType;
    import flash.text.TextField;

    public class PanningExample extends MovieClip {

        private var textField:TextField = new TextField();

        public function PanningExample() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;

            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
            textField.height = 200;
            textField.type = TextFieldType.INPUT;
            textField.border = true;
            textField.wordWrap = true;
            textField.multiline = true;

            this.addChild( textField );

            //track soft keyboard and stage resize events
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onKeyboardChange );
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onKeyboardChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
```

```
            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
        }

        private function onKeyboardChange( event:SoftKeyboardEvent ):void
        {
            var field:InteractiveObject = textField;
            var offset:int = 0;

            //if the softkeyboard is open and the field is at least partially covered
            if( (this.stage.softKeyboardRect.y != 0) && (field.y + field.height >
this.stage.softKeyboardRect.y) )
                offset = field.y + field.height - this.stage.softKeyboardRect.y;

            //but don't push the top of the field above the top of the screen
            if( field.y - offset < 0 ) offset += field.y - offset;

            this.y = -offset;
        }
    }
}
```

Obs! *På Android är tangentbordets exakta dimensioner inte tillgängliga från operativsystemet under vissa omständigheter, bl.a. i helskärmsläge. I så fall uppskattas storleken. I liggande orienteringar används dessutom det inbyggda IME-helskärmstangentbordet för all textinmatning. Det här IME-tangentbordet har ett inbyggt fält för textinmatning och döljer hela scenen. Det finns inget sätt att visa ett liggande tangentbord som inte fyller skärmen.*

Kapitel 31: Musindata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ditt program kan skapa interaktivitet genom att hämta och svara på indata från en mus. Observera att du för det här avsnittet förutsätts ha grundläggande kunskaper om ActionScript 3.0-händelsemodellen. Mer information finns i ”Hantera händelser” på sidan 125.

Se ”Identifiera indatatyper” på sidan 558 för mer information om hur du under körning identifierar vilket typ av musstöd som finns tillgängligt (ständig musmarkör, styluspenna eller pekrörelser).

Fler hjälpavsnitt

flash.ui.Mouse

flash.events.MouseEvent

”Indata från pekhändelser, gester och multitouch” på sidan 582

Samla in musindata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Musaktiviteter skapar mushändelser som kan användas för att aktivera interaktiva funktioner. Du kan lägga till en händelseavlyssnare på scenen för att lyssna efter mushändelser som inträffar i SWF-filen. Du kan också lägga händelseavlyssnare till objekt på scenen, vilka ärver från InteractiveObject (till exempel Sprite och MovieClip). Dessa avlyssnare aktiveras när någon klickar på objektet.

Mushändelser bubblar, precis som tangentbordshändelser. I nästa exempel skickas händelsen både från `square`-spriten och scenobjektet när någon klickar på fyrkanten, eftersom `square` är underordnad scenen:

```
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 + "]");
}
```

I det föregående exemplet kan du lägga märke till att mushändelsen innehåller positionell information om klickhändelsen. Egenskaperna `localX` och `localY` innehåller platsen för klickhändelsen i den lägst underordnade i visningskedjan. Genom att exempelvis klicka i det övre vänstra hörnet rapporterar `square` de lokala koordinaterna [0,0] eftersom detta är registreringspunkten för `square`. Egenskaperna `stageX` och `stageY` refererar däremot till klickningens globala koordinater på scenen. Samma klickning rapporterar [50,50] för dessa koordinater eftersom `square` flyttades till dessa koordinater. Båda dessa koordinatpar kan vara användbara beroende på hur du vill besvara användarinteraktionen.

Obs! *I helskärmsläge kan du konfigurera programmet att använda låsning av musen. Om du låser musen inaktiveras markören och obegränsade musrörelser aktiveras. Mer information finns i avsnittet "Arbeta i helskärmsläge" på sidan 167.*

Objektet MouseEvent innehåller de booleska egenskaperna `altKey`, `ctrlKey` och `shiftKey`. Du använder dessa egenskaper för att kontrollera om Alt-, Ctrl- eller skifttangenten också trycks ned samtidigt som musklickningen.

Dra och flytta sprites på scenen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan göra det möjligt för användare att dra och flytta ett Sprite-objekt på scenen med metoden `startDrag()` för Sprite-klassen. Följande kod visar ett exempel på detta:

```
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);
}
```

Mer information finns i avsnittet om att skapa musinteraktion i ”Ändra position” på sidan 173.

Dra-och-släpp i AIR

I Adobe AIR kan du aktivera stöd för dra-och-släpp-funktioner som gör det möjligt för användare att dra data in i och ut från programmet. Mer information finns i ”Dra och släppa i AIR” på sidan 609.

Anpassa muspekaren

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Muspekaren kan för alla visningsobjekt på scenen döljas eller fås att byta plats. Du döljer muspekaren genom att anropa metoden `Mouse.hide()`. Du anpassar pekaren genom att anropa `Mouse.hide()`, lyssnar av scenen efter `MouseEvent.MOUSE_MOVE`-händelsen och anger koordinaterna för ett visningsobjekt (din vanliga pekare) med händelseegenskaperna `stageX` och `stageY`. I följande exempel visas ett grundläggande exempel på hur detta kan göras:

```
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;
}
```

Exempel på indata från mus: WordSearch

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I detta exempel visas användarinteraktion med mushändelser. Användarna ska bygga så många ord som möjligt i ett slumpmässigt rutnät av bokstäver. Bokstäverna visas vågrätt eller lodrätt i rutnätet och användarna får inte använda samma bokstav två gånger. Med detta exempel visas hur du använder ActionScript 3.0 för att göra följande:

- Bygger ett dynamiskt rutnät för komponenter
- Besvarar mushändelser
- Upprätthåller poängställning baserad på användarinteraktion

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. WordSearch-programfiler finns i mappen Samples/WordSearch. Programmet består av följande filer:

Fil	Beskrivning
WordSearch.as	Klassen som innehåller de flesta funktioner för programmet.
WordSearch.fla eller WordSearch.mxml	Huvudprogramfilen för Flex (MXML) eller Flash (FLA).
dictionary.txt	En fil som används för att kontrollera om ordet är rättstavat och om det ger poäng.

Läsa in ordboken

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En ordbok behövs om du skapar ett spel där det gäller att hitta ord. I detta exempel ingår en textfil med namnet dictionary.txt som innehåller en lista med ord separerade med radmatningstecken. Sedan du skapat en array med namnet `words` begärs filen från funktionen `loadDictionary()`, och när den har lästs ändras filen till en lång sträng. Med metoden `split()` kan strängen läggas i en array med ord som bryts vid varje förekomst av ett radmatningstecken (teckenkod 10) eller tecknet för ny rad (teckenkod 13). Detta görs i funktionen `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

Skapa ett användargränssnitt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När orden har lagrats gör du inställningarna för användargränssnittet. Skapa två Button-instanser: en för att skicka ett ord och den andra för att rensa ett ord som har stavats. I båda fallen måste du svara på en användares inmatningar genom att avlyssna händelsen `MouseEvent.CLICK`, som knappen sänder, och sedan anropa en funktion. I funktionen `setupUI()` skapas avlyssnarkoden för båda knapparna:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Skapa ett spelbräde

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Spelbrädet består av ett rutnät med slumpmässiga bokstäver. Ett tvådimensionellt rutnät skapas med funktionen `generateBoard()` genom att en slinga kapslas in i en annan. Den första slingan ökar raden med ett och den andra ökar antalet kolumner per rad med ett. Varje cell som skapas av dessa rader och kolumner innehåller en knapp som representerar en bokstav på brädet.

```
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);
        }
    }
}
```

Trots att en avlyssnare lagts till på endast en rad för en `MouseEvent.CLICK`-händelse tilldelas den till varje Button-instans eftersom den finns i en `for`-slinga. Varje knapp ges dessutom ett namn som härleds för dess rad- och kolumnposition, vilket gör det lätt att senare i koden referera till raden och kolumnen för varje knapp.

Bygga upp ord från användarinmatningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ord byggs upp genom att bokstäver som ligger lodrätt eller vågrätt intill varandra markeras, men man får inte använda samma bokstav två gånger. Varje klickning genererar en mushändelse då ordet som användaren skriver kontrolleras för att se om det fortsätter från bokstäver som tidigare markerades. Om så inte är fallet tas det föregående ordet bort och ett nytt påbörjas. Den här kontrollen görs i metoden `isLegalContinuation()`.

```
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));
}
```

Metoderna `charAt()` och `indexOf()` i klassen String hämtar rätt rad och kolumn från både den senast klickade knappen och den knapp som klickades på tidigare. Metoden `isLegalContinuation()` returnerar `true` om raden eller kolumnen är oförändrad och om den inte har ändrats mer än ett steg från föregående. Om du vill ändra spelets regler och tillåta diagonala ord, tar du bort kontrollen av oförändrad rad eller kolumn. Den sista raden kommer då att få följande utseende:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

Kontrollera skickade ord

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du vill slutföra koden för spelet behövs mekanismer för att kontrollera ord som skickats och för att hålla ordning på poängen. Metoden `searchForWord()` innehåller båda dessa mekanismer:

```
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;
}
```

I funktionen gås alla ord i ordboken igenom. Om användarens ord överensstämmer med ett ord i ordboken returneras ordets position i denna. Med metoden `submitWord()` kontrolleras sedan svaren, och poängen uppdateras om positionen är giltig.

Anpassningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I början av klassen finns flera konstanter. Du kan förändra spelet genom att ändra dessa variabler. Du kan exempelvis ändra tiden för spelet genom att öka variabeln `TOTAL_TIME`. Du kan också öka variabeln `PERCENT_VOWELS` något för att öka sannolikheten för att hitta rätt ord.

Kapitel 32: Indata från pekhändelser, gester och multitouch

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Hantering av pekhändelser i Flash-plattformen innehåller funktioner för indata från en enda kontaktpunkt eller från flera kontaktpunkter på enheter som stöder pekfunktioner. Flash-körningsversionerna hanterar även händelser som kombinerar flera kontaktpunkter med rörelser, så kallade gester. Med andra ord, Flash-körningsversionerna tolkar två sorters indata:

Pek Indata med en enda kontaktenhet, till exempel ett finger, en styluspenna eller något annat verktyg på en enhet som stöder pekfunktioner. Vissa enheter har stöd för flera samtidiga kontaktpunkter med ett finger eller en styluspenna.

Multitouch Indata med flera samtidiga kontaktpunkter.

Gester Indata tolkas av en enhet eller ett operativsystem som svar på en eller flera pekhändelser. En användare kan till exempel vrida två fingrar samtidigt, vilket kan tolkas av enheten eller operativsystemet som en gest för rotering. Vissa gester utförs med endast ett finger eller en kontaktpunkt och för vissa gester krävs flera kontaktpunkter. Vilken typ av gest det är avgörs av enheten eller operativsystemet och indata tilldelas därefter.

Både pek- och geströrelser kan vara multitouch beroende på användarens enhet. ActionScript innehåller API för hantering av pekhändelser, gesthändelser och enskilt spårade pekhändelser för multitouch-indata.

Obs! *Att lyssna efter berörings- och gesthändelser kan ta stora processorresurser i anspråk (jämförbart med att återge flera bildrutor per sekund), beroende på datorenhet och operativsystem. Det är oftast bättre att använda mushändelser, om du inte absolut behöver de extrafunktioner som berörings- och gesthändelser ger. Om du ändå använder berörings- och gesthändelser bör du minska antalet grafiska förändringar som kan inträffa, särskilt vid händelser som kan skickas snabbt, som panorerings-, rotations- eller zoomningsåtgärder. Du kan till exempel stoppa animeringen i en komponent under tiden användaren ändrar storlek på den med en zoomningsgest.*

Fler hjälpavsnitt

flash.ui.Multitouch

flash.events.TouchEvent

flash.events.GestureEvent

flash.events.TransformGestureEvent

flash.events.GesturePhase

flash.events.PressAndTapGestureEvent

Paul Trani: Touch Events and Gestures on Mobile (Beröringshändelser och gester på mobiler)

Mike Jones: Virtual Game Controllers

Grunder för pekrörelser

Flash Player 10.1 och senare, Adobe AIR 2 och senare

När Flash-plattformen körs i en miljö som stöder pekrörelser kan InteractiveObject-instanser lyssna efter pekhändelser och anropa hanterare. Normalt hanteras pek-, gest- och multitouch-händelser på samma sätt som andra typer av händelser i ActionScript (se ”Hantera händelser” på sidan 125 för grundläggande information om händelsehantering med ActionScript).

Men för att Flash-körningen ska kunna tolka pekfunktioner och gester måste körningen köras i miljö med maskinvara och programvara som har stöd för pek- eller multitouch-funktioner. Se ”Identifiera indatatyper” på sidan 558 för en jämförande tabell av olika typer av pekskärmar. Om körningen körs inom ett behållarprogram (till exempel en webbläsare) skickas indata till körningen via behållaren. I vissa fall stöds multitouch i den aktuella maskinvaru- och operativsystemmiljön, men indata tolkas av webbläsaren som fungerar som behållare för Flash-körningen och skickas inte vidare till körningen. Eller så ignoreras dessa indata helt och hållet.

I följande diagram beskrivs indataflödet från användare till körning:

Indataflöde från användare till Flash-plattformkörning

Men ActionScript API för utveckling av program med pekfunktioner innehåller dess bättre klasser, metoder och egenskaper för att avgöra om det finns stöd för pek- och multitouch-funktioner i körningsmiljön. Om du vill avgöra stödet för pekfunktioner använder du ett API för identifiering av pekhändelsehantering.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer för programmering med pekhändelsehantering:

API för identifiering Metoder och egenskaper som används för att testa om körningsmiljön har stöd för pekhändelser och andra inmatningssätt.

Pekhändelse En indataåtgärd som utförs med en enda kontaktyta på en enhet med stöd för pekfunktioner.

Kontaktpunkt Kontaktpunkten för en enda pekhändelse. Även om en enhet inte har stöd för indata med hjälp av gester kan den ha stöd för flera samtidiga kontaktpunkter.

Peksekvens Den serie av händelser som tillsammans utgör en pekhändelse från början till slut. Dessa händelser utgörs av en början, en (eller inga) eller flera åtgärder samt ett slut.

Multitouch-händelse En indataåtgärd som utförs med flera kontaktpunkter (till exempel flera fingrar) på en enhet med stöd för pekfunktioner.

Gesthändelse En indataåtgärd som utförs genom att spåra ett rörelsemönster på en enhet med stöd för pekfunktioner. En gest kan till exempel vara att peka på skärmen med två fingrar och samtidigt flytta dem i en cirkel för att indikera rörelse.

Faser Avskilda tidpunkter i händelseflödet (till exempel början och slut).

Styluspenna Ett verktyg för interaktion med en pekskärm. En styluspenna har större precision än ett finger. Vissa enheter kan endast känna av indata med en särskild typ av styluspenna. Enheter som känner av indata med styluspennor kanske inte kan känna av fingerkontakt eller flera samtidiga kontaktpunkter.

Tryck-och-peka En särskild typ av gest för multitouch-indata där användaren trycker ett finger mot pekskärmen och sedan pekar på den med ett annat finger eller något annat pekdon. Den här gesten används ofta för att simulera ett högerklick med en mus i program med multitouch-funktioner.

API-struktur för pekrörelser

ActionScript API för pekrörelser har skapats för att lösa problemet med att pekrörelsehantering är beroende av Flash-körningens maskinvaru- och programvarumiljö. API för pekrörelser är huvudsakligen utformade för tre aspekter av utveckling för program med pekfunktioner: identifiering, händelser och faser. Använd dessa API tillsammans för ett mer förutsägbart beteende och bättre responsivitet för användaren, även om målenheten inte är känd vid utvecklingen av programmet.

Identifiering

Med API:et för identifiering kan du testa maskinvaru- och programvarumiljön vid körningen. De värden som anges av körningen avgör vilka pekfunktioner som är tillgängliga för Flash-körningen i den aktuella miljön. Du kan även använda en uppsättning egenskaper och metoder för identifiering om du vill att programmet ska svara på mushändelser istället för pekhändelser, om det skulle vara så att vissa pekfunktioner inte stöds i den aktuella miljön. Mer information finns i ”Identifiering för stöd av pekfunktioner” på sidan 585.

Händelser

ActionScript hanterar händelser för pekrörelser med händelseavlyssnare på samma sätt som med andra händelser. Men vid hantering av pekhändelser gäller även följande:

- En pekhändelse kan tolkas på flera olika sätt av enheten eller operativsystemet, antingen som en sekvens av flera pekhändelser, eller tillsammans, som en gest.
- Vid en enskild pekning på en pekskärm (med ett finger, en styluspenna eller annat pekdon) skickas alltid även en mushändelse. Du kan hantera mushändelsen med händelsetyperna i klassen MouseEvent. Eller så kan du utforma programmet så att det endast svarar på pekhändelser. Eller skapa ett program som svarar på både pekhändelser och mushändelser.
- Ett program kan svara på flera samtidiga pekhändelser och hantera varje händelse separat.

Normalt används API:et för identifiering för hantering av de händelser som programmet hanterar, och på vilket sätt de ska hanteras. När programmet har identifierat körningsmiljön kan det anropa den lämpliga hanteraren eller etablera ett korrekt händelseobjekt när användaren interagerar med programmet. Eller så kan programmet indikera att specifika indata inte kan hanteras i den aktuella miljön och tillhandahålla ett alternativ eller information till användaren. Mer information finns i ”Hantering av pekhändelser” på sidan 586 och ”Hantering av gesthändelser” på sidan 590.

Faser

I program med pek- och multitouch-funktioner finns objekt för pekhändelser som innehåller egenskaper för att spåra faserna av användarinteraktion. Skriv ActionScript-kod som hanterar faserna för till exempel början, uppdatering och slut av indata från användaren och som ger användaren återkoppling på dessa indata. Svara på händelsefaser genom att ändra på visuella objekt när användaren pekar på eller flyttar kontaktpunkten på skärmen. Eller använd faserna för att spåra särskilda egenskaper hos en gest samtidigt som den utförs.

För kontaktpunkthändelser spårar du hur lång tid som användaren uppehåller sig på ett visst interaktivt objekt. Ett program kan spåra flera samtidiga kontaktpunktfaser separat och hantera varje kontaktpunkt därefter.

För gester tolkar du informationen om gestens utveckling samtidigt som den utförs. Spåra koordinaterna för kontaktpunkten (eller kontaktpunkterna) samtidigt som de förflyttas på skärmen.

Identifiering för stöd av pekfunktioner

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Använd egenskaperna för klassen Multitouch om du vill ange vilken typ av pekrörelser som programmet kan hantera. Testa sedan miljön för att kontrollera att det finns stöd för de händelser som hanteras i ditt ActionScript. Etablera först och främst vilken typ av pekrörelser som programmet hanterar. Alternativen är: pekningar, gester eller inga alls (tolka alla pekrörelser som musklick och använd endast mushändelsehanterare). Använd sedan egenskaperna och metoderna för klassen Multitouch för att kontrollera att körningen körs i en miljö som stöder de pekrörelser som krävs för programmet. Testa om körningsmiljön har stöd för dessa pekrörelser (till exempel gester) och svara utifrån det.

Obs! *Egenskaperna för klassen Multitouch är statiska och hör inte till några instanser av en klass. Använd dem med syntaxen Multitouch.property, till exempel:*

```
var touchSupport:Boolean = Multitouch.supportsTouchEvents;
```

Ange indatatyp

Flash-körningen måste veta vilken typ av pekrörelser som ska tolkas, eftersom en pekhändelse kan bestå av flera olika element eller faser. Om ett finger endast pekar på en pekskärm ska körningen då skicka en pekhändelse? Eller ska den vänta på en gest? Eller ska körningen spåra pekningen som en mushändelse? Ett program som stöder pekrörelser måste etablera vilken typ av pekhändelser som ska hanteras i Flash-körningen. Använd egenskapen `Multitouch.inputMode` för att etablera typ av pekrörelser för körningen. Det kan vara något av följande tre alternativ:

Ingen Ingen särskild hantering ges för pekhändelser. Ange: `Multitouch.inputMode=MultitouchInputMode.NONE` och använd klassen MouseEvent för hantering av indata.

Pekningar Alla pekrörelser tolkas individuellt och alla kontaktpunkter kan spåras och hanteras. Ange: `Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT` och använd klassen TouchEvent för hantering av indata.

Gester Enheten eller operativsystemet tolkar indata som en serie fingerrörelser på skärmen. Enheten eller operativsystemet tilldelar rörelsen till en enda händelse av gestindata. Ange: `Multitouch.inputMode=MultitouchInputMode.GESTURE` och använder klassen TransformGestureEvent, PressAndTapGestureEvent eller GestureEvent för hantering av indata.

Se ”Hantering av pekhändelser” på sidan 586 för exempel på hur egenskapen `Multitouch.inputMode` används för att ange indatatypen innan en pekhändelse hanteras.

Test för stöd av pekrörelser

Klassen Multitouch innehåller andra egenskaper för finjustering av programmet efter det stöd för pekfunktioner som finns i den aktuella miljön. Flash-körningen anger värden för hur många kontaktpunkter som ska tillåtas samtidigt eller för vilka gester som finns tillgängliga. Om körningen körs i en miljö som inte har stöd för hantering av de pekhändelser som används i programmet tillhandahåller du användaren ett alternativ. Tillhandahåll till exempel en möjlighet för hantering av mushändelser eller ge information om vilka funktioner som är tillgängliga eller inte i den aktuella miljön.

Du kan också använda API:er för stöd av tangentbords-, pek- och mushändelser, se ”Identifiera indatatyper” på sidan 558.

Mer information om kompatibilitetstest finns i ”Felsökning” på sidan 594.

Hantering av pekhändelser

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Enkla pekhändelser hanteras på samma sätt som andra händelser, till exempel mushändelser, i ActionScript. Du kan lyssna efter en serie pekhändelser som definieras av händelsetypens konstanter i klassen TouchEvent.

Obs! *När det gäller indata med flera kontaktpunkter (till exempel när du rör en enhet med mer än ett finger) kommer den första beröringen att skicka en mushändelse och en pekhändelse.*

Så här hanterar du en enkel pekhändelse:

1. Ange att programmet ska hantera pekhändelser genom att ange egenskapen `flash.ui.Multitouch.inputMode` till `MultitouchInputMode.TOUCH_POINT`.
2. Bifoga en händelseavlyssnare till en instans av en klass som ärver egenskaperna från klassen InteractiveObject, till exempel Sprite eller TextField.
3. Ange vilken typ av pekhändelse som ska hanteras.
4. Anropa en händelsehanterare som ska göra något som svar på händelsen.

I följande exempel visas till exempel ett meddelande när någon vidrör den fyrkant som ritats på mySprite på en pekskärm:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

Egenskaper för pekhändelser

När ett fel inträffar skapas ett händelseobjekt. Objektet TouchEvent innehåller information om platsen och villkoren för en pekhändelse. Du kan använda egenskaperna för händelseobjektet för att hämta den informationen.

I följande kod skapas till exempel TouchEvent-objektet *evt*, och sedan visas egenskapen `stageX` för händelseobjektet (x-koordinaten för den punkt i Stage-området där pekningen inträffade) i textfältet:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
myTextField.text = evt.stageX.toString;
myTextField.y = 50;
addChild(myTextField);
}
```

Se klassen TouchEvent för tillgängliga egenskaper genom händelseobjektet.

Obs! *Alla TouchEvent-egenskaper stöds inte i alla körningsmiljöer. Alla enheter med pekskärm kan till exempel inte identifiera hur hårt eller löst som användaren trycker på pekskärmen. Egenskapen* `TouchEvent.pressure` *stöds därför inte på dessa enheter. Testa om olika egenskaper stöds så att du kan vara säker på att programmet fungerar som det ska. Se "Felsökning" på sidan 594 för mer information.*

Faser för pekhändelser

Spåra pekhändelser i olika nivåer över och utanför ett InteractiveObject på samma sätt som för mushändelser. Spåra också pekhändelser genom de olika faserna från början till mitten och slutet av en pekhändelse. Klassen TouchEvent ger värden för hantering av händelserna `touchBegin`, `touchMove` och `touchEnd`.

Du kan till exempel använda händelserna `touchBegin`, `touchMove` och `touchEnd` för att ge användarna visuell återkoppling när de pekar och flyttar på ett visningsobjekt:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
myTextField.width = 200;
myTextField.height = 20;
addChild(myTextField);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
function onTouchBegin(event:TouchEvent) {
    myTextField.text = "touch begin" + event.touchPointID;
}
function onTouchMove(event:TouchEvent) {
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    myTextField.text = "touch end" + event.touchPointID;
}
```

Obs! *Den initiala pekavlyssnaren finns bifogad till mySprite, men avlyssnarna för att flytta objektet och för slutet av pekhändelsen är inte bifogade. Om användarens finger eller pekdon flyttar framför visningsobjektet fortsätter scenen att lyssna efter pekhändelsen.*

Kontaktpunkts-ID

Egenskapen `TouchEvent.touchPointID` är en viktig del för att skriva program som svarar på pekrörelser. Flash-körningen tilldelar varje kontaktpunkt ett unikt `touchPointID`-värde. Kontrollera `touchPointID` innan händelsen hanteras när ett program svarar på faser eller rörelser från en pekhändelse. Indatametoderna för peka och dra i klassen Sprite använder egenskapen `touchPointID` som en parameter så att rätt indatainstans hanteras. Egenskapen `touchPointID` ser till att en händelsehanterare svarar på rätt kontaktpunkt. Annars svarar händelsehanteraren på alla instanser av pekhändelsetypen (till exempel alla `touchMove`-händelser) på enheten och skapar ett oförutsägbart beteende. Egenskapen är särskilt viktig när användaren drar och flyttar objekt.

Använd egenskapen `touchPointID` om du vill hantera en hel peksekvens. En peksekvens består av en `touchBegin`-händelse, en (eller inga) eller flera `touchMove`-händelser och en `touchEnd`-händelse, och alla har samma `touchPointID`-värde.

I följande exempel etableras ett variabelt *touchMoveID* som ett test för det korrekta `touchPointID`-värdet innan programmet svarar på en pekhändelse för att flytta ett objekt. Annars utlöses händelsehanteraren även av andra pekrörelser. Observera att avlyssnarna för flytt- och slutfaserna ligger på scenen och inte på visningsobjektet. Scenen lyssnar efter flytt- eller slutfaserna om användaren skulle flytta utanför visningsobjektets gränser.

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
addChild(myTextField);
myTextField.width = 200;
myTextField.height = 20;
var touchMoveID:int = 0;

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
function onTouchBegin(event:TouchEvent) {
    if(touchMoveID != 0) {
        myTextField.text = "already moving. ignoring new touch";
        return;
    }
    touchMoveID = event.touchPointID;

    myTextField.text = "touch begin" + event.touchPointID;
    stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
```

```
}
function onTouchMove(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch";
        return;
    }
    mySprite.x = event.stageX;
    mySprite.y = event.stageY;
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch end";
        return;
    }
    touchMoveID = 0;
    stage.removeEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.removeEventListener(TouchEvent.TOUCH_END, onTouchEnd);
    myTextField.text = "touch end" + event.touchPointID;
}
```

Peka och dra

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Två metoder har lagts till i klassen Sprite för ytterligare stöd för program med pekfunktioner som stöder pekrörelser via kontaktpunkter: `Sprite.startTouchDrag()` och `Sprite.stopTouchDrag()`. Dessa metoder har samma beteende som `Sprite.startDrag()` och `Sprite.stopDrag()` för mushändelser. Observera dock att metoderna `Sprite.startTouchDrag()` och `Sprite.stopTouchDrag()` båda tar `touchPointID`-värden som parametrar.

Körningen tilldelar `touchPointID`-värdet till händelseobjektet för en pekhändelse. Använd det här värdet som svar på en specifik kontaktpunkt om miljön har stöd för flera samtidiga kontaktpunkter (även om den inte hanterar gester). Mer information om egenskapen `touchPointID` finns i ”Kontaktpunkts-ID” på sidan 588.

I följande kod visas en enkel händelsehanterare för händelsen börja dra och en händelsehanterare för händelsen sluta dra. Variabeln *bg* är ett visningsobjekt som innehåller *mySprite*:

```
mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(e:TouchEvent) {
    e.target.startTouchDrag(e.touchPointID, false, bg.getRect(this));
    trace("touch begin");
}

function onTouchEnd(e:TouchEvent) {
    e.target.stopTouchDrag(e.touchPointID);
    trace("touch end");
}
```

Och nedan visas ett mer avancerat exempel som kombinerar dra-händelsen med faser för pekhändelsen:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(evt:TouchEvent) {
    evt.target.startTouchDrag(evt.touchPointID);
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}

function onTouchMove(evt:TouchEvent) {
    evt.target.alpha = 0.5;
}

function onTouchEnd(evt:TouchEvent) {
    evt.target.stopTouchDrag(evt.touchPointID);
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
```

Hantering av gesthändelser

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Gesthändelser hanteras på samma sätt som enkla pekhändelser. Du kan lyssna efter en serie gesthändelser som definieras av konstanterna för händelsetypen i klassen TransformGestureEvent, GestureEvent och PressAndTapGestureEvent.

Så här hanterar du en pekhändelse med gester:

1. Ange att programmet ska hantera gester genom att ange egenskapen `flash.ui.Multitouch.inputMode` till `MultitouchInputMode.GESTURE`.
2. Bifoga en händelseavlyssnare till en instans av en klass som ärver egenskaperna från klassen InteractiveObject, till exempel Sprite eller TextField.
3. Ange vilken typ av gesthändelse som ska hanteras.
4. Anropa en händelsehanterare som ska göra något som svar på händelsen.

I följande exempel visas till exempel ett meddelande när någon sveper vid den fyrkant som ritats på *mySprite* på en pekskärm:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, swipehandler);

function swipehandler(evt:TransformGestureEvent): void {
myTextField.text = "I've been swiped";
myTextField.y = 50;
addChild(myTextField);
}
```

Pekhändelser som baseras på två fingrar hanteras på samma sätt, men använder klassen GestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, taphandler);

function taphandler(evt:GestureEvent): void {
myTextField.text = "I've been two-finger tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

Tryck-och-peka-händelser hanteras också på samma sätt, men använder klassen PressAndTapGestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP, taphandler);

function taphandler(evt:PressAndTapGestureEvent): void {
myTextField.text = "I've been press-and-tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

Obs! *Alla GestureEvent-, TransformGestureEvent- och PressAndTapGestureEvent-hänselsetyper stöds inte i alla körningsmiljöer. Alla enheter med pekskärm kan till exempel inte identifiera svepningar med flera fingrar. Händelserna* `gestureSwipe` *för InteractiveObject stöds därför inte på dessa enheter. Testa om olika händelser stöds så att du kan vara säker på att programmet fungerar som det ska. Se "Felsökning" på sidan 594 för mer information.*

Egenskaper för gesthändelser

Gesthändelser har färre händelseegenskaper än enkla pekhändelser. Du får tillgång till dem på samma sätt, via händelseobjektet för händelsehanteraren.

I följande kod roteras till exempel *mySprite* när användaren utför en rotationsgest på den. I textfältet visas den grad av rotation som använts sedan den senaste gesten (vid testning av koden ser du till att rotera den flera gånger för att se om värdena ändras):

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var mySpriteCon:Sprite = new Sprite();
var myTextField:TextField = new TextField();
myTextField.y = 50;
addChild(myTextField);

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(-20,-20,40,40);
mySpriteCon.addChild(mySprite);
mySprite.x = 20;
mySprite.y = 20;
addChild(mySpriteCon);

mySprite.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rothandler);

function rothandler(evt:TransformGestureEvent): void {
evt.target.parent.rotationZ += evt.target.rotation;
myTextField.text = evt.target.parent.rotation.toString();
}
```

Obs! *Alla TransformGestureEvent-egenskaper stöds inte i alla körningsmiljöer. Alla enheter med pekskärm kan till exempel inte identifiera en gestrotation på skärmen. Egenskapen* `TransformGestureEvent.rotation` *stöds därför inte på dessa enheter. Testa om olika egenskaper stöds så att du kan vara säker på att programmet fungerar som det ska. Se "Felsökning" på sidan 594 för mer information.*

Faser för gesthändelser

Gesthändelser kan dessutom spåras genom olika faser så att du kan spåra egenskaper samtidigt som geströrelsen utförs. Du kan till exempel spåra x-koordinater samtidigt som ett objekt flyttas med en svepningsgest. Använd dessa värden om du vill rita en linje genom alla punkter i banan efter att svepningen har slutförts. Eller ändra ett visningsobjekt visuellt samtidigt som det dras över skärmen med en panoreringsgest. Ändra objektet igen när panoreringsgesten har slutförts.

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {

    evt.target.localX++;

if (evt.phase==GesturePhase.BEGIN) {
    myTextField.text = "Begin";
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}
if (evt.phase==GesturePhase.UPDATE) {
    myTextField.text = "Update";
    evt.target.alpha = 0.5;
}
if (evt.phase==GesturePhase.END) {
    myTextField.text = "End";
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
}
```

Obs! *Intervallet för uppdateringsfasen beror på körningsmiljön. I vissa kombinationer av operativsystem och maskinvara skickas inte några uppdateringar ut alls.*

Gestfasen är ”alla” för enkla gesthändelser

Vissa objekt för gesthändelser spårar inte enskilda faser av gesthändelsen, utan istället anges värdet ”alla” för händelseobjektets fasegenskap. För de enkla gesterna svepning och tvåfingerpek spåras inte händelsen genom flera faser. Egenskapen `phase` för händelseobjektet för ett InteractiveObject som lyssnar efter `gestureSwipe`- eller `gestureTwoFingerTap`-händelser är alltid `all` när händelsen väl har skickats:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, onSwipe);
mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, onTwoTap);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onSwipe(swipeEvt:TransformGestureEvent):void {
    myTextField.text = swipeEvt.phase // Output is "all"
}
function onTwoTap(tapEvt:GestureEvent):void {
    myTextField.text = tapEvt.phase // Output is "all"
}
```

Felsökning

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Stödet för olika pekfunktioner i maskinvara och program ändras mycket snabbt. Den här referensen innehåller inte någon lista över varje enhet eller kombination av operativsystem och programvara som stöder multitouch. Men den ger en vägledning i hur du använder API:et för identifiering för att avgöra om programmet är implementerat på en enhet som har stöd för multitouch, och ger tips på hur du felsöker din ActionScript-kod.

Flash-körningar svarar på pekhändelser utifrån den information som enheten, operativsystemet eller programbehållaren (till exempel en webbläsare) skickar till körningen. Detta beroendeförhållandet till programmiljön komplicerar beskrivningen av multitouch-kompatibiliteten. Vissa enheter tolkar en gest eller en pekrörelse annorlunda än andra enheter. Ska ett objekt roteras genom att två fingrar vrids på samma gång? Ska det roteras genom att ett finger gör en cirkelrörelse på skärmen? Beroende på vilken maskinvaru- och programmiljö som används kan gesten för att rotera objekt se annorlunda ut. Enheten ger därför operativsystemet information om användarens indata och operativsystemet skickar detta vidare till körningen. Om körningen sker i en webbläsare, tolkar webbläsareprogrammet ibland gesten eller pekhändelsen och skickar den inte vidare till körningen. Det här beteendet liknar beteendet med tangentbordskommandon: om du försöker att använda en viss tangentkombination för Flash Player i en webbläsare kanske webbläsaren istället öppnar en meny.

Enskilda API:er och klasser ger information om de inte är kompatibla med ett visst operativsystem. Du kan utforska de enskilda API:erna här, med början från klassen Multitouch: http://help.adobe.com/sv_SE/FlashPlatform/reference/actionscript/3/flash/ui/Multitouch.html.

Här är en beskrivning av några vanliga gester och pekrörelser:

Panorera Flytta ett finger från vänster till höger eller från höger till vänster. På vissa enheter krävs två fingrar för att panorera.

Rotera Trycka ned två fingrar och sedan flytta runt dem i en cirkel (som om fingrarna ritade en cirkel på ytan). Pivåpunkten är mittpunkten mellan de båda fingrarnas kontaktpunkter.

Svepa Snabbt flytta tre fingrar från vänster till höger eller från höger till vänster, eller uppifrån och ned eller nerifrån och upp.

Zooma Håll ned två fingrar och flytta dem sedan ifrån varandra om du vill zooma in och närmare varandra om du vill zooma ut.

Tryck-och-peka Flytta eller tryck ned ett finger och peka sedan på ytan med ett annat.

Varje enhet har egen dokumentation om de gester som den enheten stöder och hur gesterna utförs på enheten. Oftast måste användaren ta bort alla fingrar från enheten mellan olika gester beroende på vilket operativsystem som används.

Om programmet inte svarar på pekhändelser eller gester testar du följande:

1. Finns händelseavlyssnare för pek- eller gesthändelser bifogade till en objektklass som ärver från klassen InteractiveObject? Endast InteractiveObject-instanser kan lyssna efter pek- och gesthändelser
2. Testar du programmet i Flash Professional CS5? Försök i så fall publicera och testa programmet eftersom Flash Professional kan störa interaktionen.
3. Börja enkelt och se vad som fungerar först (följande kodexempel är från API:et för `Multitouch.inputMode`:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField()

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taplistener);

function taplistener(e:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

   Peka på rektangeln Om det här exemplet fungerar så vet du att din miljö stöder enkla pekningar. Sedan kan du försöka med en mer komplicerad hantering.

   Tester för geststöd är mer komplicerade. Varje enskild enhet eller operativsystem har antingen stöd för gestkombinationer eller inte.

   Här visas en enkel test för zoomgester:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_ZOOM , onZoom);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a zoom gesture";
addChild(myTextField);

function onZoom(evt:TransformGestureEvent):void {
    myTextField.text = "Zoom is supported";
}
```

   Utför en zoomgest på enheten och kontrollera om fältet fylls i med meddelandet `Zoom is supported`. Händelseavlyssnaren läggs till på scenen så att du kan utföra gesten på alla delar i testprogrammet.

   Här visas en enkel test för panoreringsgester:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a pan gesture";
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {
    myTextField.text = "Pan is supported";
}
```

Utför en panoreringsgest på enheten och kontrollera om fältet fylls i med meddelandet `Pan is supported`. Händelseavlyssnaren läggs till på scenen så att du kan utföra gesten på alla delar i testprogrammet.

Vissa operativsystem och enhetskombinationer har stöd för båda gesterna, vissa har stöd för endast en och andra ingen. Kontrollera programmets distributionsmiljö för att vara säker.

Kända fel

Nedan följer kända problem som rör beröringsindata:

1. Mobile Internet Explorer på operativsystemet Windows Mobile zoomar automatiskt in SWF-filinnehåll:

   Det här zoombeteendet i Internet Explorer kan undvikas genom att du lägger till följande kod i HTML-sidan som hyser SWF-filen:

   ```
   <head>
   <meta name="viewport" content="width=device-width, height=device-height, initial-
   scale=1.0">
   </head>
   ```

2. Windows 7 (och eventuellt andra operativsystem): Användaren måste lyfta på pekdonet (eller fingrarna) från skärmen mellan olika gester. Så här gör du till exempel för att rotera och zooma en bild:
   - Utför roteringsgesten.
   - Lyft bort fingrarna från skärmen.
   - Sätt tillbaka fingrarna på skärmen och utför zoomgesten.
3. Windows 7 (och eventuellt andra operativsystem): Rotations- och zoomningsgesterna genererar inte alltid någon uppdateringsfas om användaren utför gesten mycket snabbt.
4. Windows 7 Starter Edition stöder inte multitouch. Mer information finns i forumet för AIR Labs: http://forums.adobe.com/thread/579180?tstart=0
5. I Mac OS 10.5.3 och senare är värdet för `Multitouch.supportsGestureEvents` alltid `true`, även om maskinvaran inte har stöd för gesthändelser.

Kapitel 33: Kopiera och klistra in

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Använd klasserna i urklipps-API:et när du kopierar information till och från systemets Urklipp. De dataformat som kan överföras till eller från ett program som körs i Adobe® Flash® Player eller Adobe® AIR® är bland annat följande:

- Text
- HTML-formaterad text
- RTF-data
- Serialiserade objekt
- Objektreferenser (endast giltiga inom originalprogrammet)
- Bitmappar (enbart AIR)
- Filer (enbart AIR)
- URL-strängar (endast AIR)

Grunderna om kopiera och klistra in

Flash Player 10 och senare, Adobe AIR 1.0 och senare

API:et för kopiera och klistra in innehåller följande klasser.

Paket	Klasser
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

Den statiska egenskapen `Clipboard.generalClipboard` representerar operativsystemets Urklipp. Klassen Clipboard innehåller metoder för att läsa och skriva data till urklippsobjekt.

Klassen HTMLLoader (i AIR) och klassen TextField implementerar standardbeteendet för de normala kortkommandona för kopiera och klistra in. Om du vill implementera kortkommandobeteenden för kopiera och klistra in för egna komponenter, kan du avlyssna dessa tangenttryckningar direkt. Du kan också använda inbyggda menykommandon tillsammans med tangentbordsmotsvarigheter om du vill reagera på tangenttryckningar direkt.

Olika representationer av samma information kan göras tillgängliga i ett enda Clipboard-objekt om du vill öka möjligheterna för andra program att förstå och använda data. En bild kan till exempel inkluderas som bilddata, ett serialiserat Bitmap-objekt och som en fil. Återgivningen av data i ett format kan fördröjas så att formatet inte skapas förrän data i det formatet läses.

Läsa från och skriva till systemets Urklipp

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Om du vill läsa operativsystemets Urklipp anropar du metoden `getData()` för objektet `Clipboard.generalClipboard` och skickar namnet på formatet som ska läsas:

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

Obs! *Innehåll som körs i Flash Player eller i en icke-programsandlåda i AIR kan bara anropa metoden `getData()` i en händelsehanterare för en `paste`-händelse. Det är alltså bara kod som körs i AIR:s programsandlåda som kan anropa metoden `getData()` utanför en `paste`-händelsehanterare.*

Om du vill skriva till Urklipp lägger du till data i objektet `Clipboard.generalClipboard` i ett eller flera format. Alla befintliga data i samma format skrivs över automatiskt. Det är trots det en bra rutin att även rensa systemets Urklipp innan nya data skrivs dit, så att också orelaterade data i alla format tas bort.

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

Obs! *Innehåll som körs i Flash Player eller i en icke-programsandlåda i AIR kan bara anropa metoden `setData()` i en händelsehanterare för en användarhändelse, till exempel en tangentbords- eller mushändelse eller en `copy`- eller `cut`-händelse. Det är alltså bara kod som körs i AIR:s programsandlåda som kan anropa metoden `setData()` utanför en användarhändelsehanterare.*

HTML-kopiering och -inklistring i AIR

Adobe AIR 1.0 och senare

HTML-miljön i Adobe AIR har en egen uppsättning med händelser och standardbeteenden för att kopiera och klistra in. Det är bara kod som körs i sandlådan application som har direkt åtkomst till systemets Urklipp via AIR-objektet `Clipboard.generalClipboard`. JavaScript-kod i en annan typ av sandlåda än application kan få åtkomst till Urklipp via händelseobjektet som skickas som svar på någon av copy- eller paste-händelserna som skickas av ett element i ett HTML-dokument.

Kopiera- och klistra in-händelser omfattar: `copy`, `cut` och `paste`. Objektet som skickas för dessa händelser ger åtkomst till Urklipp via egenskapen `clipboardData`.

Standardbeteende

Adobe AIR 1.0 och senare

AIR kopierar som standard markerade objekt som svar på kopieringskommandot, som kan skapas med ett kortkommando eller en snabbmeny. I redigerbara områden klipper AIR ut text som svar på utklippskommandot eller klistrar in text vid markören eller markeringen som svar på klistra in-kommandot.

Om du inte vill använda standardbeteendet kan händelsehanteraren anropa metoden `preventDefault()` för det skickade händelseobjektet.

Använda egenskapen clipboardData för händelseobjektet

Adobe AIR 1.0 och senare

Egenskapen `clipboardData` för händelseobjektet som skickas som ett resultat av någon av copy- eller paste-händelserna gör det möjligt att läsa och skriva Urklipps-data.

Om du vill skriva till Urklipp vid en copy- eller cut-händelse ska du använda metoden `setData()` för objektet `clipboardData`, så skickar du de data som ska kopieras samt MIME-typen:

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

Om du vill ha åtkomst till de data som klistras in kan du använda metoden `getData()` för objektet `clipboardData`, som skickas i MIME-typen för dataformatet. Vilka format som finns rapporteras i egenskapen `types`.

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

Metoden `getData()` och egenskapen `types` är bara tillgängliga i händelseobjektet som skickas av händelsen `paste`.

I exemplet nedan visas hur du åsidosätter standardbeteendet för kopiera och klistra in på en HTML-sida. Hanteraren för händelsen `copy` gör den kopierade texten kursiv och kopierar den till Urklipp som HTML-text. Hanteraren för händelsen `cut` kopierar markerade data till Urklipp och tar bort dem från dokumentet. Hanteraren för händelsen `paste` infogar innehållet i Urklipp som HTML och formaterar infogningen med fetstil.

```
 <html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

Format för Clipboard-data

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Clipboard-formaten beskriver vilka data som placeras i ett Clipboard-objekt. Flash Player eller AIR översätter automatiskt standarddataformaten mellan ActionScript-datatyper och systemets urklippsformat. Programobjekt kan dessutom överföras inom och mellan ActionScript-baserade program med hjälp av programdefinierade format.

Ett Clipboard-objekt kan innehålla representationer av samma information i olika format. Ett Clipboard-objekt som till exempel representerar en Sprite kan omfatta ett referensformat som ska användas i samma program, ett serialiserat format som ska användas av ett annat program som kör Flash Player eller AIR, ett bitmappsformat som ska användas i en bildredigerare och ett fillisteformat, kanske med fördröjd återgivning för kodning av en PNG-fil, för att kopiera och dra en representation av denna Sprite till filsystemet.

Standarddataformat

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Konstanterna som definierar standardformatnamnen finns i klassen ClipboardFormats:

Konstant	Beskrivning
TEXT_FORMAT	Textformatdata översätts till och från ActionScript-klassen String.
HTML_FORMAT	Text med HTML-kod.
RICH_TEXT_FORMAT	RTF-data översätts till och från ActionScript-klassen ByteArray. RTF-koden varken tolkas eller översätts på något sätt.
BITMAP_FORMAT	(Enbart AIR) Bitmappsdata översätts till och från ActionScript-klassen BitmapData.
FILE_LIST_FORMAT	(Enbart AIR) Data i fillisteformat översätts till och från en array med ActionScript File-objekt.
URL_FORMAT	(Enbart AIR) URL-formatdata översätts till och från ActionScript-klassen String.

När du vill kopiera och klistra in data som svar på en `copy`-, `cut`- eller `paste`-händelse i HTML-innehåll i ett AIR-program måste MIME-typer användas i stället för ClipboardFormat-strängar. Giltiga MIME-typer för data är:

MIME-typ	Beskrivning
Text	”text/plain”
URL	”text/uri-list”
Bitmapp	”image/x-vnd.adobe.air.bitmap”
Fillista	”application/x-vnd.adobe.air.file-list”

Obs! *RTF-data är inte tillgängliga från egenskapen `clipboardData` för händelseobjektet som skickas som ett resultat av en `paste`-händelse i HTML-innehåll.*

Egna dataformat

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Du kan använda programdefinierade anpassade format om du vill överföra objekt som referenser eller som serialiserade objekt. Referenser är bara giltiga inom samma program. Serialiserade objekt kan överföras mellan program, men kan bara användas med objekt som förblir giltiga vid serialisering och deserialisering. Objekt kan vanligtvis serialiseras om egenskaperna är enkla typer eller serialiserbara objekt.

Om du vill lägga till ett serialiserat objekt i ett Clipboard-objekt ställer du in parametern *serializable* på `true` när du anropar metoden `Clipboard.setData()`. Formatnamnet kan vara något av standardformaten eller en godtycklig sträng som definieras i programmet.

Överföringslägen

Flash Player 10 och senare, Adobe AIR 1.0 och senare

När ett objekt skrivs till Urklipp med hjälp av ett anpassat dataformat, kan objektdata läsas från Urklipp antingen som en referens eller som en serialiserad kopia av det ursprungliga objektet. Det finns fyra överföringslägen som avgör om objekten överförs som referenser eller som serialiserade kopior:

Överföringsläge	Beskrivning
ClipboardTransferModes.ORIGINAL_ONLY	Enbart en referens returneras. Om ingen referens finns, returneras ett null-värde.
ClipboardTransferModes.ORIGINAL_PREFFERED	En referens returneras, om en sådan finns. Annars returneras en serialiserad kopia.
ClipboardTransferModes.CLONE_ONLY	Enbart en serialiserad kopia returneras. Om ingen serialiserad kopia finns returneras ett null-värde.
ClipboardTransferModes.CLONE_PREFFERED	En serialiserad kopia returneras, om en sådan finns. Annars returneras en referens.

Läsa och skriva anpassade dataformat

Flash Player 10 och senare, Adobe AIR 1.0 och senare

När ett objekt skrivs till Urklipp kan du använda valfri sträng som inte börjar med något av de reserverade prefixen `air:` eller `flash:` för parametern *format*. Använd samma sträng som formatet som ska läsa objektet. I följande exempel visas hur objekt läses och skrivs till Urklipp:

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Om du vill extrahera ett serialiserat objekt från urklippsobjektet (efter en åtgärd av typen släppa eller klistra in) använder du samma formatnamn och överföringsläget `CLONE_ONLY` eller `CLONE_PREFFERED`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

En referens läggs alltid till i objektet Clipboard. Om du vill extrahera referensen från urklippsobjektet (efter en åtgärd av typen släppa eller klistra in), i stället för en serialiserad kopia, använder du överföringsläget `ORIGINAL_ONLY` eller `ORIGINAL_PREFFERED`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Referenserna är bara giltiga om Clipboard-objektet härstammar från det aktuella programmet. Använd överföringsläget `ORIGINAL_PREFFERED` om du vill få tillgång till referensen när den är tillgänglig och till den serialiserade klonen när referensen inte är tillgänglig.

Fördröjd återgivning

Flash Player 10 och senare, Adobe AIR 1.0 och senare

Om det tar mycket datorkraft att skapa ett dataformat, kan du använda fördröjd återgivning genom att ange en funktion som tillhandahåller data på begäran. Funktionen anropas bara om en mottagare av en släpp- eller klistra in-åtgärd begär data i det fördröjda formatet.

Återgivningsfunktionen läggs till i ett Clipboard-objekt med hjälp av metoden `setDataHandler()`. Funktionen måste returnera data i lämpligt format. Om du till exempel anropar `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)` måste funktionen `writeText()` returnera en sträng.

Om ett dataformat av samma typ läggs till i ett Clipboard-objekt med metoden `setData()`, prioriteras dessa data över den fördröjda versionen (återgivningsfunktionen anropas aldrig). Återgivningsfunktionen anropas eventuellt inte igen om samma urklippsdata hämtas en andra gång.

Obs! *I Mac OS X fungerar fördröjd återgivning bara med anpassade dataformat. Med standarddataformat anropas återgivningsfunktionen omedelbart.*

Klistra in text med funktionen för fördröjd återgivning

Flash Player 10 och senare, Adobe AIR 1.0 och senare

I följande exempel visas hur du implementerar en funktion för fördröjd återgivning.

När användaren klickar på knappen Kopiera, raderas systemets Urklipp så att inga data finns kvar från föregående urklippsåtgärder. Metoden `setDataHandler()` ställer sedan in funktionen `renderData()` som urklippsåtergivning.

När användaren väljer kommandot Klistra in på snabbmenyn i måltextfältet får programmet tillgång till urklipp och ställer in måltexten. Eftersom textdataformatet i Urklipp har ställts in med en funktion i stället för en sträng, anropar Urklipp funktionen `renderData()`. Funktionen `renderData()` returnerar texten i källtexten, som sedan tilldelas måltexten.

Observera, att om du redigerar källtexten innan du trycker på knappen Klistra in, återspeglas redigeringen i den inklistrade texten även om redigeringen görs efter att knappen Kopiera trycks ned. Detta beror på att återgivningsfunktionen inte kopierar källtexten förrän knappen Klistra in trycks ned. (När fördröjd återgivning används i ett riktigt program, kanske du bör lagra eller skydda källdata på något sätt för att undvika det här problemet.)

Flash-exempel

```
package {
   import flash.desktop.Clipboard;
   import flash.desktop.ClipboardFormats;
   import flash.desktop.ClipboardTransferMode;
   import flash.display.Sprite;
   import flash.text.TextField;
   import flash.text.TextFormat;
   import flash.text.TextFieldType;
   import flash.events.MouseEvent;
   import flash.events.Event;
   public class DeferredRenderingExample extends Sprite
   {
       private var sourceTextField:TextField;
       private var destination:TextField;
       private var copyText:TextField;
       public function DeferredRenderingExample():void
       {
           sourceTextField = createTextField(10, 10, 380, 90);
           sourceTextField.text = "Neque porro quisquam est qui dolorem "
               + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

           copyText = createTextField(10, 110, 35, 20);
           copyText.htmlText = "<a href='#'>Copy</a>";
           copyText.addEventListener(MouseEvent.CLICK, onCopy);

           destination = createTextField(10, 145, 380, 90);
           destination.addEventListener(Event.PASTE, onPaste);
       }
       private function createTextField(x:Number, y:Number, width:Number,
                       height:Number):TextField
       {
           var newTxt:TextField = new TextField();
           newTxt.x = x;
           newTxt.y = y;
           newTxt.height = height;
           newTxt.width = width;
```

```
            newTxt.border = true;
            newTxt.multiline = true;
            newTxt.wordWrap = true;
            newTxt.type = TextFieldType.INPUT;
            addChild(newTxt);
            return newTxt;
        }
        public function onCopy(event:MouseEvent):void
        {
            Clipboard.generalClipboard.clear();
            Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                            renderData);
        }
        public function onPaste(event:Event):void
        {
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Flex-exempel

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326"
height="330" applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text =
Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur,
adipisci velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

Kapitel 34: Indata från accelerometer

Flash Player 10.1 och senare, Adobe AIR 2 och senare

Klassen Accelerometer skickar händelser baserat på aktivitet som identifieras av enhetens rörelsesensor. Dessa data representerar enhetens placering eller förflyttning längs en tredimensionell axel. När enheten rör sig känner sensorn av rörelsen och returnerar enhetens accelerationskoordinater. Klassen Accelerometer innehåller metoder för att fråga om accelerometerfunktionen stöds samt för att ange den hastighet med vilken accelerationshändelser skickas.

Accelerometeraxlarna normaliseras efter visningsorienteringen, inte enhetens fysiska orientering. När enheten orienterar om skärmen orienteras även accelerometeraxlarna om. Y-axeln är således alltid ungefär lodrät när användaren håller telefonen i ett normalt, upprätt visningsläge, oavsett åt vilket håll telefonen har roterats. Om automatisk orientering är inaktiverat, och SWF-innehåll i webbläsaren visas i helskärmsläge, orienteras accelerometeraxlarna inte om när enheten roteras.

Fler hjälpavsnitt

flash.sensors.Accelerometer

flash.events.AccelerometerEvent

Kontrollera stöd för accelerometern

Använd egenskapen `Accelerometer.isSupported` för att testa om körningsmiljön kan använda den här funktionen:

```
if (Accelerometer.isSupported)
{
    // Set up Accelerometer event listeners and code.
}
```

Klassen Accelerometer med dess medlemmar är tillgängliga i de körningsversioner som visas för varje API. Men det är den aktuella miljön vid körningen som avgör om funktionen är tillgänglig eller inte. Du kan till exempel kompilera kod med egenskaperna för klassen Accelerometer för Flash Player 10.1, men du behöver använda egenskapen `Accelerometer.isSupported` för att testa om accelerometerfunktionen är tillgänglig på användarens enhet. Om värdet för `Accelerometer.isSupported` är `true` vid körningen stöds accelerometerfunktionen.

Identifiera accelerometerändringar

Använd sensorn för accelerometern genom att instansiera ett Accelerometer-objekt och registrera `update`-händelser som skickas. Händelsen `update` är ett Accelerometer-händelseobjekt. Händelsen har fyra egenskaper, som anges med siffror:

- `accelerationX` – Accelerationen längs med x-axeln mätt i antal g:n. X-axeln går från vänster till höger på enheten när den står upprätt. (Enheten är i upprätt läge när den övre delen av enheten är pekar uppåt.) Accelerationen är positiv om enheten flyttas åt höger.

- `accelerationX` – Accelerationen längs med y-axeln mätt i antal g:n. Y-axeln går nedifrån och upp på enheten när den står upprätt. (Enheten är i upprätt läge när den övre delen av enheten är pekar uppåt.) Accelerationen är positiv om enheten flyttas uppåt i förhållande till axeln.
- `accelerationZ` – Accelerationen längs med z-axeln mätt i antal g:n. Z-axeln går vinkelrätt mot enhetens kant. Accelerationen är positiv om enheten flyttas så att ytan pekar uppåt. Accelerationen är negativ om ytan på enheten pekar mot marken.
- `timestamp` – Det antal millisekunder som vid tiden för händelsen har passerat sedan körningen initierades.

1 g är standardaccelerationen på grund av gravitationen, cirka 9,8 m/sek^{2}.

Det här grundläggande exemplet visar accelerometerdata i ett textfält:

```
var accl:Accelerometer;
if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(evt:AccelerometerEvent):void
{
    accTextField.text = "acceleration X: " + evt.accelerationX.toString() + "\n"
            + "acceleration Y: " + evt.accelerationY.toString() + "\n"
            + "acceleration Z: " + evt.accelerationZ.toString()
}
```

Om du vill använda exemplet måste du först skapa textfältet `accTextField` och lägga till det i visningslistan innan koden används.

Du kan justera det önskade tidsintervallet för accelerometerhändelser genom att anropa metoden `setRequestedUpdateInterval()` för Accelerometer-objektet. Den här metoden tar en parameter, `interval`, vilken är det uppdateringsintervall som krävs i millisekunder:

```
var accl:Accelerometer;
accl = new Accelerometer();
accl.setRequestedUpdateInterval(1000);
```

Den faktiska tiden mellan uppdateringarna av accelerometern kan vara längre eller kortare än det här värdet. Förändringar av uppdateringsintervallet påverkar alla registrerade avlyssnare. Om du inte anropar metoden `setRequestedUpdateInterval()` tar programmet emot uppdateringar beroende på enhetens standardintervall.

Accelerometerdata kan ha en viss grad av inexakthet. Du kan använda ett glidande medelvärde av senaste data för utjämning av data. I följande exempel faktoreras de senaste accelerometerläsningarna med den aktuella läsningen för ett mer avrundat resultat:

```
var accl:Accelerometer;
var rollingX:Number = 0;
var rollingY:Number = 0;
var rollingZ:Number = 0;
const FACTOR:Number = 0.25;

if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.setRequestedUpdateInterval(200);
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(event:AccelerometerEvent):void
{
    accelRollingAvg(event);
    accTextField.text = rollingX + "\n" +  rollingY + "\n" + rollingZ + "\n";
}

function accelRollingAvg(event:AccelerometerEvent):void
{
    rollingX = (event.accelerationX * FACTOR) + (rollingX * (1 - FACTOR));
    rollingY = (event.accelerationY * FACTOR) + (rollingY * (1 - FACTOR));
    rollingZ = (event.accelerationZ * FACTOR) + (rollingZ * (1 - FACTOR));
}
```

Men detta glidande medelvärde är endast önskvärt om uppdateringsintervallet för accelerometern är litet.

Kapitel 35: Dra och släppa i AIR

Adobe AIR 1.0 och senare

Använd klasserna i API:t för dra och släpp i Adobe® AIR™ om du vill kunna använda dra och släpp-gester i ett användargränssnitt. En *gest* i det här fallet är en användaråtgärd, som skickas både av operativsystemet och programmet, och uttrycker en vilja att kopiera, flytta eller länka information. En *utdragningsgest* sker när användaren drar ett objekt ut från en komponent eller ett program. En *indragningsgest* sker när användaren drar in ett objekt från utsidan av en komponent eller ett program.

Med API:et för dra och släpp kan du se till att en användare kan dra mellan program och mellan komponenter inom ett program. Bland annat följande överföringsformat stöds:

- Bitmappar
- Filer
- HTML-formaterad text
- Text
- RTF-data
- URL:er
- Fillöften
- Serialiserade objekt
- Objektreferenser (endast giltiga inom originalprogrammet)

Grunderna i dra och släpp i AIR

Adobe AIR 1.0 och senare

Följande snabbstartartiklar hos Adobe Developer Connection innehåller korta förklaringar och kodexempel på hur du använder dra och släpp i ett AIR-program:

- Stöd för dra-och-släpp och kopiera-och-klistra in (Flex)
- Stöd för dra-och-släpp och kopiera-och-klistra in (Flash)

API:et för dra och släpp innehåller följande klasser.

Paket	Klasser
flash.desktop	• NativeDragManager • NativeDragOptions • Clipboard • URLFilePromise • IFilePromise Konstanterna som används med API:et för dra och släpp definieras i följande klasser: • NativeDragActions • ClipboardFormat • ClipboardTransferModes
flash.events	NativeDragEvent

Gestfaser för dra och släpp

Dra och släpp-gesten har tre faser:

Initiering *En användare initierar en dra och släpp-åtgärd genom att dra från en komponent, eller ett objekt i en komponent, samtidigt som han eller hon håller ned musknappen.* Komponenten som utgör källan för objektet som dras betecknas vanligtvis som draginitieraren och skickar händelserna `nativeDragStart` och `nativeDragComplete`. Ett Adobe AIR-program startar en dra-åtgärd genom att anropa metoden `NativeDragManager.doDrag()` som svar på en `mouseDown`- eller `mouseMove`-händelse.

Om dra-åtgärden påbörjas utanför ett AIR-program finns det inget initieringsobjekt som skickar `nativeDragStart`- eller `nativeDragComplete`-händelser.

Dra *När användaren håller ned musknappen flyttas musmarkören till en annan komponent, till ett program eller till skrivbordet.* Så länge dragningen pågår skickar initieringsobjektet `nativeDragUpdate`-händelserna. (Den här händelsen skickas emellertid inte i AIR för Linux.) När användaren flyttar musen över ett möjligt släppmål i ett AIR-program skickar släppmålet en `nativeDragEnter`-händelse. Händelsehanteraren kan granska händelseobjektet för att ta reda på om de data som dras är tillgängliga i ett format som accepteras på målet och, om så är fallet, låta användare släppa data på det genom att anropa metoden `NativeDragManager.acceptDragDrop()`.

Så länge dragningsgesten finns kvar över ett interaktivt objekt skickar objektet `nativeDragOver`-händelser. När dragningsgesten lämnar det interaktiva objektet skickas en `nativeDragExit`-händelse.

Släppa *Användaren släpper musknappen över ett möjligt släppmål.* Om målet är ett AIR-program eller en AIR-komponent kommer målobjektet att skicka en `nativeDragDrop`-händelse. Händelsehanteraren kan få tillgång till den överförda datan från händelseobjektet. Om målet finns utanför AIR hanterar operativsystemet eller andra programhanterare släppningen. I båda fallen skickar initieringsobjektet en `nativeDragComplete`-händelse (om dragningen börjar från AIR).

Klassen NativeDragManager kontrollerar både indragnings- och utdragningsgester. Alla medlemmar i klassen NativeDragManager är statiska, så du ska inte skapa en instans av den här klassen.

Objektet Clipboard

Data som dras in till eller ut från ett program eller en komponent som finns i ett Clipboard-objekt. Ett enda Clipboard-objekt kan göra olika representationer tillgängliga av samma information så att chanserna ökar att ett annat program kan förstå och använda dessa data. En bild kan till exempel inkluderas som bilddata, ett serialiserat Bitmap-objekt och som en fil. Återgivningen av data i ett format kan skjutas upp till en återgivningsfunktion som inte anropas förrän informationen läses.

När en dragningsgest har startat kan objektet Clipboard bara nås inifrån en händelsehanterare för händelserna `nativeDragEnter`, `nativeDragOver` och `nativeDragDrop`. När dragningsgesten är färdig kan Clipboard-objektet inte längre läsas eller återanvändas.

Ett programobjekt kan överföras som en referens och som ett serialiserat objekt. Referenser är endast giltiga inom originalprogrammet. Serialiserade objekt kan överföras mellan AIR-program, men kan bara användas med objekt som förblir giltiga när det serialiseras och deserialiseras. Objekt som serialiseras konverteras till Action Message Format för ActionScript 3 (AMF3), ett strängbaserat dataöverföringsformat.

Arbeta med Flex-ramverket

I de flesta fall är det bättre att använda API:et för dra och släpp i Adobe® Flex™ när du bygger Flex-program. Flex-ramverket har motsvarande funktioner när ett Flex-program körs i AIR (det använder AIR NativeDragManager internt). Flex upprätthåller också begränsade funktioner när ett program eller en komponent körs i den mer restriktiva webbläsarmiljön. AIR-klasser kan inte användas i komponenter eller program som körs utanför AIR-körningsversionen.

Stöd för utdragningsgesten

Adobe AIR 1.0 och senare

Om du vill ha stöd för utdragningsgesten måste du skapa ett Clipboard-objekt som svar på en `mouseDown`-händelse och skicka det till metoden `NativeDragManager.doDrag()`. Programmet kan sedan avlyssna händelsen `nativeDragComplete` för initieringsobjektet för att ta reda på vilken åtgärd som ska utföras när användaren slutför eller avbryter gesten.

Förbereda data för överföring

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du förbereder data eller ett objekt för dragning genom att skapa ett Clipboard-objekt och lägga till den information som ska överföras i ett eller flera format. Du kan använda standarddataformaten för att skicka data, som kan översättas automatiskt till inbyggda urklippsformat, och programdefinierade format för att skicka objekt.

Om det är resurskrävande att konvertera den information som ska överföras till ett visst format kan du ange namnet på en hanterarfunktion som utför konverteringen. Funktionen anropas bara om mottagande komponent eller program läser det tillhörande formatet.

Mer information om urklippsformat finns i ”Format för Clipboard-data” på sidan 600.

I följande exempel visas hur du skapar ett Clipboard-objekt som innehåller en bitmapp i flera format: ett Bitmap-objekt, ett inbyggt bitmappsformat och ett fillisteformat som innehåller filen som bitmappen ursprungligen lästes in från:

```
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
public function createClipboard(image:Bitmap, sourceFile:File):Clipboard{
    var transfer:Clipboard = new Clipboard();
    transfer.setData("CUSTOM_BITMAP", image, true); //Flash object by value and by reference
    transfer.setData(ClipboardFormats.BITMAP_FORMAT, image.bitmapData, false);
    transfer.setData(ClipboardFormats.FILE_LIST_FORMAT, new Array(sourceFile), false);
    return transfer;
}
```

Starta en utdragningsåtgärd

Adobe AIR 1.0 och senare

Du startar en dragningsåtgärd genom att anropa metoden `NativeDragManager.doDrag()` som svar på en mouseDown-händelse. Metoden `doDrag()` är en statisk metod som använder följande parametrar:

Parameter	Beskrivning
initiator	Objektet som dragningen härstammar från, och som skickar händelserna `dragStart` och `dragComplete`. Initieraren måste vara ett interaktivt objekt.
clipboard	Clipboard-objektet innehåller data som ska överföras. Clipboard-objektet refereras i NativeDragEvent-objekten som skickas under dra och släpp-sekvensen.
dragImage	(Valfritt) Ett BitmapData-objekt som ska visas under dragningen. Bilden kan ange ett `alfa`-värde. (Obs! Microsoft Windows använder alltid en fast alfatoning när bilder dras.)
offset	(Valfritt) Ett Point-objekt som anger förskjutningen för dragningsbilden från musens aktiva punkt. Använd negativa koordinater om du vill flytta bilden upp eller åt vänster i förhållande till musmarkören. Om ingen förskjutning anges, placeras dragningsbildens övre, vänstra hörn vid musens aktiva punkt.
actionsAllowed	(Valfritt) Ett NativeDragOptions-objekt som anger vilka åtgärder (kopiera, flytta eller länka) som kan användas för dragningsåtgärden. Om inga argument anges tillåts alla åtgärder. DragOptions-objektet refereras i objektet NativeDragEvent, vilket gör det möjligt för ett eventuellt dragmål att kontrollera att tillåtna åtgärder är kompatibla med målkomponentens syfte. En ”papperskorgskomponent” kanske bara accepterar dragningsgester som tillåter åtgärden flytta.

I följande exempel visas hur du startar en dragningsåtgärd för ett bitmappsobjekt som har lästs in från en fil. Exemplet läser in en bild och, vid en `mouseDown`-händelse, startar en dragningsåtgärd.

```
package
{
import flash.desktop.NativeDragManager;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.display.Loader;
import flash.system.LoaderContext;
import flash.net.URLRequest;
import flash.geom.Point;
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
import flash.events.Event;
import flash.events.MouseEvent;

public class DragOutExample extends UIComponent Sprite {
    protected var fileURL:String = "app:/image.jpg";
    protected var display:Bitmap;

    private function init():void {
        loadImage();
    }
    private function onMouseDown(event:MouseEvent):void {
        var bitmapFile:File = new File(fileURL);
        var transferObject:Clipboard = createClipboard(display, bitmapFile);
        NativeDragManager.doDrag(this,
                            transferObject,
                            display.bitmapData,
                            new Point(-mouseX,-mouseY));
    }
    public function createClipboard(image:Bitmap, sourceFile:File):Clipboard {
        var transfer:Clipboard = new Clipboard();
        transfer.setData("bitmap",
                            image,
                            true);
                            // ActionScript 3 Bitmap object by value and by reference
        transfer.setData(ClipboardFormats.BITMAP_FORMAT,
                            image.bitmapData,
                            false);
                            // Standard BitmapData format
        transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
```

```
                            new Array(sourceFile),
                            false);
                            // Standard file list format
        return transfer;
    }
    private function loadImage():void {
        var url:URLRequest = new URLRequest(fileURL);
        var loader:Loader = new Loader();
        loader.load(url,new LoaderContext());
        loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onLoadComplete);
    }
    private function onLoadComplete(event:Event):void {
        display = event.target.loader.content;
        var flexWrapper:UIComponent = new UIComponent();
        flexWrapper.addChild(event.target.loader.content);
        addChild(flexWrapper);
        flexWrapper.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    }
}
}
```

Slutföra utdragningsöverföringen

Adobe AIR 1.0 och senare

När en användare släpper det dragna objektet genom att släppa upp musknappen skickar initieringsobjektet en `nativeDragComplete`-händelse. Du kan kontrollera egenskapen `dropAction` för händelseobjektet och sedan vidta rätt åtgärd. Om åtgärden till exempel är `NativeDragAction.MOVE` kan du ta bort källobjektet från dess ursprungliga plats. Användaren kan avbryta en dragningsgest genom att släppa upp musknappen medan markören finns utanför ett möjligt släppmål. Draghanteraren ställer in egenskapen `dropAction` för en avbruten gest på `NativeDragAction.NONE`.

Stöd för indragningsgesten

Adobe AIR 1.0 och senare

Om du vill ha stöd för indragningsgesten måste programmet (eller för det mesta en visuell komponent i programmet) svara på `nativeDragEnter`- eller `nativeDragOver`-händelser.

Steg i en vanlig släppåtgärd

Adobe AIR 1.0 och senare

Följande sekvens med händelser brukar ingå i en släppåtgärd:

1. Användaren drar ett urklippsobjekt över en komponent.
2. Komponenten skickar en `nativeDragEnter`-händelse.
3. Hanteraren för händelsen `nativeDragEnter` undersöker händelseobjektet för att kontrollera tillgängliga dataformat och tillåtna åtgärder. Om komponenten kan hantera släppningen anropas `NativeDragManager.acceptDragDrop()`.

4 NativeDragManager ändrar musmarkören för att ange att objektet kan släppas.

5 Användaren släpper objektet över komponenten.

6 Den mottagande komponenten skickar en `nativeDragDrop`-händelse.

7 Den mottagande komponenten läser data i önskat format från Clipboard-objektet i händelseobjektet.

8 Om dragningsgesten härstammar från ett AIR-program skickar det initierande interaktiva objektet en `nativeDragComplete`-händelse. Om gesten började utanför AIR skickas ingen feedback.

Bekräfta en indragningsgest

Adobe AIR 1.0 och senare

När en användare drar ett urklippsobjekt innanför gränserna för en visuell komponent skickas händelserna `nativeDragEnter` och `nativeDragOver`. Om du vill ta reda på om komponenten accepterar urklippsobjektet kan hanterarna för dessa händelser kontrollera egenskaperna `clipboard` och `allowedActions` för händelseobjektet. Om du vill signalera att komponenten kan acceptera släppningen måste händelsehanteraren anropa metoden `NativeDragManager.acceptDragDrop()`, och skicka en referens till den mottagande komponenten. Om mer än en registrerad händelseavlyssnare anropar metoden `acceptDragDrop()` har den sista hanteraren i listan högsta prioritet. `acceptDragDrop()`-anropet fortsätter att gälla tills musen lämnar gränserna för det accepterande objektet, och händelsen `nativeDragExit` utlöses.

Om mer än en åtgärd tillåts i parametern `allowedActions` som skickas till `doDrag()`, kan användaren ange vilken av de tillåtna åtgärderna han eller hon tänker utföra genom att hålla ned en modifieringstangent. Draghanteraren ändrar markörbilden för att ange för användaren vilken åtgärd som sker om han eller hon slutför släppningen. Den tänkta åtgärden rapporteras i egenskapen `dropAction` för NativeDragEvent-objektet. Åtgärderna för en dragningsgest är bara förslag. Komponenterna som ingår i överföringen måste implementera rätt beteende. En flyttåtgärd slutförs till exempel genom att dragningsinitieraren tar bort det dragna objektet och släppmålet lägger till det.

Dragmålet kan begränsa släppåtgärden till en av tre möjliga åtgärder genom att du ställer in egenskapen `dropAction` för klassen NativeDragManager. Om en användare försöker att välja en annan åtgärd med tangentbordet visar NativeDragManager markören *otillgänglig*. Ställ in egenskapen `dropAction` i hanterarna för både `nativeDragEnter`- och `nativeDragOver`-händelserna.

I följande exempel visas en händelsehanterare för en `nativeDragEnter`- eller `nativeDragOver`-händelse. Den här hanteraren accepterar bara en indragningsgest om urklippet som dras innehåller data i textformat.

```
import flash.desktop.NativeDragManager;
import flash.events.NativeDragEvent;

public function onDragIn(event:NativeDragEvent):void{
    NativeDragManager.dropAction = NativeDragActions.MOVE;
    if(event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
        NativeDragManager.acceptDragDrop(this); //'this' is the receiving component
    }
}
```

Slutföra släppningen

Adobe AIR 1.0 och senare

När användaren släpper ett draget objekt i ett interaktivt objekt som har accepterat gesten, skickar det interaktiva objektet en `nativeDragDrop`-händelse. Hanteraren för den här händelsen kan extrahera data från egenskapen `clipboard` för händelseobjektet.

När urklippet innehåller ett programdefinierat format, bestämmer parametern `transferMode` som skickas till metoden `getData()` för Clipboard-objektet om draghanteraren returnerar en referens eller en serialiserad version av objektet.

I följande exempel visas en händelsehanterare för händelsen `nativeDragDrop`:

```
 import flash.desktop.Clipboard;
import flash.events.NativeDragEvent;

public function onDrop(event:NativeDragEvent):void {
    if (event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)) {
    var text:String =
        String(event.clipboard.getData(ClipboardFormats.TEXT_FORMAT,
                                    ClipboardTransferMode.ORIGINAL_PREFERRED));
}
```

När händelsehanteraren avslutas gäller inte längre Clipboard-objektet. Alla försök att få åtkomst till objektet eller dess data genererar ett fel.

Uppdatera en komponents visuella utseende

Adobe AIR 1.0 och senare

En komponent kan uppdatera sitt visuella utseende baserat på NativeDragEvent-händelserna. I tabellen nedan beskrivs vilka typer av ändringar som en vanlig komponent kan göra som svar på olika händelser:

Händelse	Beskrivning
nativeDragStart	Det initierande interaktiva objektet kan använda händelsen `nativeDragStart` för att ge visuell feedback till dragningsgesten som härstammar från det interaktiva objektet.
nativeDragUpdate	Det initierande interaktiva objektet kan använda händelsen nativeDragUpdate för att uppdatera sin status under gesten. (Den här händelsen finns inte i AIR för Linux.)
nativeDragEnter	Ett eventuellt mottagande objekt kan använda den här händelsen för att blir markerat, eller ange visuellt att det kan eller inte kan acceptera släppningen.
nativeDragOver	Ett eventuellt mottagande interaktivt objekt kan använda den här händelsen för att svara på musens rörelser inuti det interaktiva objektet, till exempel när musen kommer till ett "aktivt" område i en komplex komponent som en stadskarta.
nativeDragExit	Ett eventuellt mottagande objekt kan använda den här händelsen för att återställa sitt läge när en dragningsgest flyttar utanför dess gränser.
nativeDragComplete	Det initierande interaktiva objektet kan använda den här händelsen för att uppdatera den tillhörande datamodellen, till exempel genom att ta bort ett objekt från en lista, och återställa sitt visuella läge.

Spåra musens placering under en indragningsgest

Adobe AIR 1.0 och senare

När en dragningsgest stannar kvar över en komponent skickas `nativeDragOver`-händelser. Dessa händelser skickas med ett par millisekunders mellanrum och även när musen flyttas. `nativeDragOver`-händelseobjektet kan användas för att ta reda på musens placering över komponenten. Att ha tillgång till muspositionen kan vara praktiskt när den mottagande komponenten är komplex, men inte utgörs av underkomponenter. Om programmet till exempel visar en bitmapp som innehåller en stadskarta, och du vill markera områden på kartan när användaren drar information till dem, kan du använda musens koordinater som rapporteras i händelsen `nativeDragOver` för att spåra musens placering på kartan.

Dra och släpp i HTML

Adobe AIR 1.0 och senare

Om du vill dra data till och ut från ett HTML-baserat program (eller till och ut från den HTML som visas i HTMLLoader), kan du använda dra och släpp-händelserna för HTML. Med API:et för dra och släppa i HTML kan du dra till och från DOM-element i HTML-innehållet.

Obs! *Du kan också använda AIR-API:erna NativeDragEvent och NativeDragManager genom att avlyssna händelser för HTMLLoader-objektet som rymmer HTML-innehållet. HTML-API:t är emellertid bättre integrerat med HTML DOM och ger dig kontroll över standardbeteendet.*

Standardbeteende för dra och släpp

Adobe AIR 1.0 och senare

HTML-miljön innehåller standardbeteenden för dra och släpp-gester som omfattar text, bilder och URL:er. Med hjälp av standardbeteendet kan du alltid dra dessa typer av data ut från ett element. Du kan emellertid bara dra text till ett element och bara till element i ett redigerbart område på en sida. När du drar text mellan eller inom redigerbara områden på en sida utför standardbeteendet en flyttåtgärd. När du drar texten till ett redigerbart område från ett icke redigerbart område eller från utsidan av ett program, utför standardbeteendet en kopieringsåtgärd.

Du kan åsidosätta standardbeteendet genom att hantera dra och släpp-händelserna själv. Om du vill avbryta standardbeteendet måste du anropa `preventDefault()`-metoderna för objekten som skickas för dra och släpp-händelserna. Du kan sedan infoga data i släppmålet och ta bort data från dragningskällan om det behövs för att utföra den valda åtgärden.

Användaren kan som standard dra och släppa all text samt dra bilder och länka. Med WebKit CSS-egenskapen, `-webkit-user-select`, kan du kontrollera hur HTML-element markeras. Om du till exempel ställer in `-webkit-user-select` på `none` kan elementinnehållet inte markeras och därför inte dras. Du kan också använda CSS-egenskapen `-webkit-user-drag` för att kontrollera om ett element som helhet kan dras. Innehållet i elementet behandlas emellertid separat. Användaren kan fortfarande dra en markerad del av texten. Mer information finns i ”CSS i AIR” på sidan 980.

Dra och släpp-händelser i HTML

Adobe AIR 1.0 och senare

Händelserna som skickas av initierarelement som dragningen startar från är följande:

Händelse	Beskrivning
dragstart	Skickas när användaren inleder dragningsgesten. Hanteraren för den här händelsen kan förhindra dragningen om det behövs, genom att anropa metoden preventDefault() för händelseobjektet. Om du vill styra hur den dragna informationen kan kopieras, länkas eller flyttas ställer du in egenskapen effectAllowed. Markerad text, bilder och länkar placeras i Urklipp av standardbeteendet, men du kan ställa in andra data för dragningsgesten via egenskapen dataTransfer för händelseobjektet.
drag	Skickas fortlöpande under dragningsgesten.
dragend	Skickas när användaren släpper upp musknappen för att avsluta dragningsgesten.

Följande händelser skickas av ett dragningsmål:

Händelse	Beskrivning
dragover	Skickas fortlöpande medan dragningsgesten finns kvar inom elementets gränser. Hanteraren för den här händelsen bör ställa in egenskapen dataTransfer.dropEffect till att ange om släppningen leder till en kopierings-, flytt- eller länkningsåtgärd om användaren släpper musknappen.
dragenter	Skickas när någon drar innanför gränsen till ett element. Om du ändrar några egenskaper för ett dataTransfer-objekt i en hanterare för händelsen dragenter, åsidosätts dessa ändringar snabbt av nästa dragover-händelse. Å andra sidan sker det en kort fördröjning mellan en dragenter-händelse och den första dragover-händelsen som kan göra så att markören blinkar om olika egenskaper har ställts in. I många fall kan du använda samma händelsehanterare för båda händelserna.
dragleave	Skickas när dragningsgesten lämnar elementets gränser.
drop	Skickas när användaren släpper data i elementet. Data som dras kan bara nås inom hanteraren för den här händelsen.

Händelseobjektet som skickas som svar på dessa händelser påminner om en mushändelse. Du kan använda mushändelseegenskaperna, till exempel (`clientX`, `clientY`) och (`screenX`, `screenY`), för att ta reda på musens placering.

Den viktigaste händelsen för ett dragningshändelseobjekt är `dataTransfer`, som innehåller de data som dras. Objektet `dataTransfer` har följande egenskaper och metoder:

Egenskap eller metod	Beskrivning
effectAllowed	Effekten som tillåts av källan till dragningen. Hanteraren för händelsen dragstart ställer vanligtvis in detta värde. Se "Dragningseffekter i HTML." på sidan 619.
dropEffect	Effekten som väljs av målet eller användaren. Om du ställer in `dropEffect` i en `dragover`- eller `dragenter`-händelsehanterare uppdaterar AIR musmarkören till att ange vilken effekt som sker om användaren släpper upp musknappen. Om `dropEffect`-inställningen inte motsvarar någon av de tillåtna effekterna, går det inte att släppa och markören *otillgänglig* visas. Om du inte har ställt in en `dropEffect` som svar på den senaste `dragover`- eller `dragenter`-händelsen kan användaren välja från de tillåtna effekterna med operativsystemets vanliga modifieringstangenter. Den slutliga effekten rapporteras av egenskapen `dropEffect` för objektet som skickas för `dragend`. Om användaren avbryter släppningen genom att släppa upp musknappen utanför ett möjligt mål, ställs `dropEffect` in på `none`.
types	En array som innehåller MIME-typsträngar för varje dataformat som anges i objektet `dataTransfer`.

Egenskap eller metod	Beskrivning
getData(mimeType)	Hämtar data i det format som anges i parametern `mimeType`. Metoden `getData()` kan bara anropas som svar på händelsen `drop`.
setData(mimeType)	Lägger till data i `dataTransfer` i det format som anges med parametern mimeType. Du kan lägga till data i flera format genom att anropa `setData()` för varje MIME-typ. Alla data som placeras i objektet `dataTransfer` med standarddragningsbeteendet rensas. Metoden `setData()` kan bara anropas som svar på händelsen `dragstart`.
clearData(mimeType)	Raderar alla data i det format som anges i parametern `mimeType`.
setDragImage(image, offsetX, offsetY)	Ställer in en anpassad dragningsbild. Metoden `setDragImage()` kan bara anropas som svar på händelsen dragstart och bara när ett helt HTML-element dras genom att dess CSS-format `-webkit-user-drag` anges som `element`. Parametern `image` kan vara ett JavaScript Element- eller Image-objekt.

MIME-typer för dra och släpp i HTML

Adobe AIR 1.0 och senare

MIME-typerna som används med `dataTransfer`-objektet för en dra och släpp-händelse i HTML omfattar:

Dataformat	MIME-typ
Text	”text/plain”
HTML	”text/html”
URL	”text/uri-list”
Bitmapp	”image/x-vnd.adobe.air.bitmap”
Fillista	”application/x-vnd.adobe.air.file-list”

Du kan använda andra MIME-strängar, bland annat programdefinierade strängar. Emellertid kanske andra program inte känner igen eller kan använda den överförda informationen. Det är ditt ansvar att lägga till data i objektet `dataTransfer` i det förväntade formatet.

Viktigt! *Det är bara kod som körs i sandlådan application som har åtkomst till släppta filer. Ett försök att läsa eller ställa in egenskaper för ett File-objekt i en annan sandlåda än application returnerar ett säkerhetsfel. Mer information finns i* *”Hantera filsläppningar i HTML-sandlådor av annan typ än application” på sidan 624.*

Dragningseffekter i HTML.

Adobe AIR 1.0 och senare

Initieraren till dragningsgesten kan begränsa vilka dragningseffekter som tillåts genom att ställa in egenskapen `dataTransfer.effectAllowed` i hanteraren för händelsen `dragstart`. Följande strängvärden kan användas:

Strängvärde	Beskrivning
"none"	Inga dragningsåtgärder tillåts.
”copy”	Data kopieras till målet, medan originalet finns kvar.
”link”	Data delas med släppmålet med hjälp av en länk tillbaka till originalet.
”move”	Data kopieras till målet och tas bort från den ursprungliga platsen.

Strängvärde	Beskrivning
"copyLink"	Data kan kopieras eller länkas.
"copyMove"	Data kan kopieras eller flyttas.
"linkMove"	Data kan länkas eller flyttas.
"all"	Data kan kopieras, flyttas eller länkas. *All* är standardeffekten när du förhindrar standardbeteendet.

Målet för dragningsgesten kan ställa in egenskapen `dataTransfer.dropEffect` till att ange vilken åtgärd som ska vidtas om användaren slutför släppningen. Om släppeffekten är en av de tillåtna åtgärderna, visas lämplig kopierings-, flyttnings- eller länkningsmarkör. Om inte, visas markören *otillgänglig*. Om ingen släppningseffekt ställs in av målet, kan användaren välja mellan de tillåtna åtgärderna med hjälp av modifieringstangenterna.

Ställ in värdet `dropEffect` i hanterarna för både `dragover`- och `dragenter`-händelserna:

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

***Obs!** Även om du alltid bör ställa in egenskapen `dropEffect` i hanteraren för `dragenter`, ska du vara medveten om att nästa `dragover`-händelser återställer egenskapen till standardvärdet. Ställ in `dropEffect` som svar på båda händelserna.*

Dra data ut från ett HTML-element

Adobe AIR 1.0 och senare

Standardbeteendet tillåter att det mesta innehållet på en HTML-sida kopieras genom släppning. Du kan kontrollera vilket innehåll som tillåts att dras med hjälp av CSS-egenskaperna `-webkit-user-select` och `-webkit-user-drag`.

Åsidosätt standardbeteendet för utdragning i hanteraren för händelsen `dragstart`. Anropa metoden `setData()` för egenskapen `dataTransfer` för händelseobjektet om du vill placera egna data i dragningsgesten.

Om du vill ange vilken dragningsgest som ett källobjekt stöder när du inte använder dig av standardbeteendet, ställer du in egenskapen `dataTransfer.effectAllowed` för händelseobjektet som skickas till händelsen `dragstart`. Du kan välja valfri kombination av effekter. Om ett källelement stöder både effekten *copy* och *link* ställer du in egenskapen på `"copyLink"`.

Ställa in data som har dragits

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Lägg till data för dragningsgesten i hanteraren för händelsen `dragstart` med egenskapen `dataTransfer`. Använd metoden `dataTransfer.setData()` om du vill placera data i Urklipp, och skicka MIME-typen och data som ska överföras.

Om du till exempel har ett bildelement i programmet, med ID *imageOfGeorge*, använder du följande dragstart-händelsehanterare. Det här exemplet lägger till representationer av en bild på George i flera dataformat, som ökar chanserna för att andra program ska kunna använda de data som dras.

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                               new Array(dragFile));
}
```

***Obs!** När du anropar metoden `setData()` för objektet `dataTransfer` läggs inga data till med standardbeteendet för dra och släpp.*

Dra data till ett HTML-element

Adobe AIR 1.0 och senare

Standardbeteendet tillåter bara att text dras till redigerbara områden på sidan. Du kan ange att ett element och dess underordnade ska göras redigerbara genom att ta med attributet `contenteditable` i elementets inledande kod. Du kan också göra hela dokumentet redigerbart genom att ställa in dokumentobjektets `designMode`-egenskap på `"on"`.

Du kan använda ett annat indragningsbeteende på en sida genom att hantera händelserna `dragenter`, `dragover` och `drop` för alla element som accepterar data som dras.

Aktivera indragning

Adobe AIR 1.0 och senare

För att du ska kunna hantera indragningsgesten måste du först avbryta standardbeteendet. Avlyssna händelserna `dragenter` och `dragover` för alla HTML-element som du vill använda som släppmål. I hanterarna för dessa händelser anropar du metoden `preventDefault()` för det skickade händelseobjektet. Om du avbryter standardbeteendet kan icke redigerbara områden ta emot en släppning.

Hämta släppta data

Adobe AIR 1.0 och senare

Du har åtkomst till den släppta datan i hanteraren för händelsen `ondrop`:

```
function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

Använd metoden `dataTransfer.getData()` om du vill läsa data i Urklipp, och skicka MIME-typen för det dataformat som ska läsas. Du tar reda på vilka dataformat som är tillgängliga med egenskapen `types` för objektet `dataTransfer`. Arrayen `types` innehåller MIME-typsträngen för respektive tillgängligt format.

När du avbryter standardbeteendet för händelsen dragenter eller dragover, ansvarar du för att infoga alla släppta data på rätt plats i dokumentet. Det finns inget API för att konvertera muspositionen till en insättningspunkt i ett element. Den här begränsningen kan göra det svårt att implementera dragningsgester av infogningstyp.

Exempel: Åsidosätta standardbeteendet för indragning i HTML

Adobe AIR 1.0 och senare

Det här exemplet implementerar ett släppmål som visar en tabell med alla dataformat som finns i det släppta objektet.

Standardbeteendet som används för att tillåta att text, länkar och bilder dras inom programmet. Exemplet åsidosätter standardbeteendet för indragning för div-elementet som fungerar som släppmål. Nyckelsteget när du gör det möjligt för icke redigerbart innehåll att acceptera en indragningsgest är att anropa metoden `preventDefault()` för händelseobjektet, som skickas för både händelsen `dragenter` och `dragover`. Som svar på en `drop`-händelse konverterar hanteraren den överförda informationen till ett HTML-radelement och infogar raden till en tabell för visning.

```
<html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }

    function dragEnterOverHandler(event){
```

```
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -
1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-
vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
       <p>Items to drag:</p>
    <ul id="source">
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
        Uses "-webkit-user-drag:none" style.
           </li>
```

```
            <li style="-webkit-user-select:none;">
         Uses "-webkit-user-select:none" style.
            </li>

        </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
        <p>Drag items from the source list (or elsewhere).</p>
        <table id="displayTable" border="1">
         <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap
Data</th></tr>
            <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </
td></tr>
        </table>
        </div>
</div>
</body>
</html>
```

Hantera filsläppningar i HTML-sandlådor av annan typ än application

Adobe AIR 1.0 och senare

Innehåll i sandlådor av annan typ än application har inte tillgång till File-objekt som skapas när filer dras till ett AIR-program. Det går heller inte att skicka något av dessa File-objekt till programinnehåll via en sandlådebrygga. (Objektegenskaperna måste nås under serialisering.) Du kan emellertid fortfarande släppa filer i programmet genom att avlyssna nativeDragDrop-händelser för objektet HTMLLoader i AIR.

Om en användare släpper en fil i en bildruta som är värd för icke programbaserat innehåll, sprids vanligtvis inte släpphändelsen från den underordnade till den överordnade. Eftersom händelserna skickas av HTMLLoader (som är behållare för allt HTML-innehåll i ett AIR-program) som inte tillhör HTML-händelseflödet, kan du emellertid ändå ta emot släpphändelsen i programinnehållet.

För att kunna ta emot händelsen för ett filsläpp, lägger det överordnade dokumentet till en händelseavlyssnare i HTMLLoader-objektet med hjälp av referensen i `window.htmlLoader`:

```
 window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

Följande exempel använder ett överordnat dokument som läser in en underordnad sida till en fjärrsandlåda (http://localhost/). Det överordnade avlyssnar händelsen `nativeDragDrop` för HTMLLoader-objektet och skriver ut filens URL.

```
 <html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

Det underordnade dokumentet måste presentera ett giltigt släppmål genom att anropa Event-objektets `preventDefault()`-metod i HTML-händelsehanterarna `dragenter` och `dragover`. Annars kan släpphändelsen inte utföras.

```
 <html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

Släppa fillöften

Adobe AIR 2 och senare

Ett fillöfte är ett urklippsformat för dra och släpp. Med ett sådant kan en användare dra ut en fil som fortfarande inte finns ur ett AIR-program. Med fillöften kan till exempel programmet tillåta en användare att dra en proxyikon till en mapp på skrivbordet. Proxyikonen representerar en fil eller vissa data som ska finnas på en URL. När användaren släpper ikonen hämtas data av körningen och filen skrivs till den plats där ikonen släpptes.

Du kan använda klassen URLFilePromise i ett AIR-program om du vill dra och släppa filer som finns på en URL. URLFilePromise-implementeringen tillhandahålls i aircore-biblioteket som en del av AIR 2 SDK. Använd antingen filen aircore.swc eller aircore.swf som hittas i SDK-katalogen frameworks/libs/air.

Alternativt kan du implementera en egen fillöfteslogik genom att använda IFilePromise-gränssnittet (som definieras i flash.desktop-paketet för körningen).

Fillöften påminner om fördröjd återgivning med en datahanterarfunktion i Urklipp. Använd fillöften i stället för fördröjd återgivning när du drar och släpper filer. Tekniken med fördröjd återgivning kan leda till ofrivilliga pauser i dra-gesten när data skapas eller hämtas. Använd fördröjd återgivning för åtgärderna kopiera och klistra in (fillöften stöds inte för dem).

Begränsningar när fillöften används

För fillöften gäller följande begränsningar jämfört med andra dataformat som du kan lägga i ett Urklipp för dra och släpp:

- Fillöften kan endast dras ut ur ett AIR-program. De kan inte släppas i ett AIR-program.
- Fillöften kan inte användas i alla operativsystem. Använd egenskapen `Clipboard.supportsFilePromise` för att kontrollera om det finns stöd för fillöften på värdsystemet. I system som saknar stöd för fillöften får du ange en alternativ mekanism för att hämta eller skapa fildata.
- Fillöften kan inte användas med Urklipp för kopiera och klistra in (`Clipboard.generalClipboard`).

Fler hjälpavsnitt

flash.desktop.IFilePromise

air.desktop.URLFilePromise

Släppa fjärrfiler

Adobe AIR 2 och senare

Använd klassen URLFilePromise om du vill skapa fillöftesobjekt som representerar filer eller data som finns på en URL. Lägg till ett eller flera fillöftesobjekt i Urklipp med hjälp av urklippsformatet `FILE_PROMISE_LIST`. I följande exempel hämtas en enskild fil och sparas som bar.txt på platsen där den släpps. Filen finns på http://www.example.com/foo.txt. (Namnet på fjärrfilen behöver inte stämma överens med namnet på den lokala filen.)

```
if( Clipboard.supportsFilePromise )
{
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/foo.txt");
    filePromise.relativePath = "bar.txt";

    var fileList:Array = new Array( filePromise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

Du kan tillåta användaren att dra mer än en fil i taget genom att lägga till fler fillöftesobjekt i den array som är kopplad till Urklipp. Du kan också ange underkataloger i egenskapen `relativePath` så att vissa eller alla filer som omfattas av åtgärden placeras i en undermapp beroende på var de släpps.

Följande exempel visar hur en dra-åtgärd som omfattar flera fillöften ska initieras. I det här exemplet placeras HTML-sidan *article.html* i Urklipp som ett fillöfte tillsammans med två länkade bildfiler. Bilderna kopieras till undermappen *images* för att relativa länkar ska bevaras.

```
if( Clipboard.supportsFilePromise )
{   //Create the promise objects
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/article.html");
    filePromise.relativePath = "article.html";

    var image1Promise:URLFilePromise = new URLFilePromise();
    image1Promise.request = new URLRequest("http://example.com/images/img_1.jpg");
    image1Promise.relativePath = "images/img_1.html";
    var image2Promise:URLFilePromise = new URLFilePromise();
    image2Promise.request = new URLRequest("http://example.com/images/img_2.jpg");
    image2Promise.relativePath = "images/img_2.jpg";


    //Put the promise objects onto the clipboard inside an array
    var fileList:Array = new Array( filePromise, image1Promise, image2Promise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    //Start the drag operation
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

Implementera gränssnittet IFilePromise

Adobe AIR 2 och senare

Om du vill ange fillöften för resurser som inte kan kommas åt med hjälp av ett URLFilePromise-objekt kan du implementera gränssnittet IFilePromise i en anpassad klass. Gränssnittet IFilePromise anger vilka metoder och egenskaper som används av AIR-körningen för att komma åt data som ska skrivas till en fil när fillöften släpps.

En IFilePromise-implementering skickar ett annat objekt till AIR-körningen som tillhandahåller data för fillöftet. Objektet måste implementera gränssnittet IDataInput som används av AIR-körningen för att läsa data. Klassen URLFilePromise som implementerar IFilePromise använder till exempel ett URLStream-objekt som dataleverantör.

I AIR kan data läsas synkront eller asynkront. IFilePromise-implementeringen rapporterar vilket åtkomstläge som stöds genom att returnera korrekt värde i egenskapen `isAsync`. Om asynkron dataåtkomst tillhandahålls måste dataleverantörsobjektet implementera gränssnittet IEventDispatcher och skicka nödvändiga händelser som `open`, `progress` och `complete`.

Som dataleverantör för ett fillöfte kan du använda en anpassad klass, eller någon av följande inbyggda klasser:

- ByteArray (synkron)
- FileStream (synkron eller asynkron)
- Socket (asynkron)
- URLStream (asynkron)

Om du vill implementera gränssnittet IFilePromise måste du ange kod för följande funktioner och egenskaper:

- `open():IDataInput` – Returnerar det dataleverantörsobjekt från vilket data läses för fillöftet. Objektet måste implementera gränssnittet IDataInput. Om data tillhandahålls asynkront måste objektet också implementera gränssnittet IEventDispatcher och skicka nödvändiga händelser (se ”Använda en asynkron dataleverantör i ett fillöfte” på sidan 629).

- `get relativePath():String` – Anger sökvägen, inklusive filnamn, för den skapade filen. Sökvägen skapas i relation till den plats där användaren väljer att släppa objektet i åtgärden dra och släpp. Om du vill vara säker på att rätt avgränsningstecken för värdoperativsystemet används i sökvägen använder du konstanten `File.separator` när du anger sökvägar som innehåller kataloger. Du kan lägga till en inställningsmetod eller använda en konstruktorparameter för att tillåta att sökvägen ställs in under körning.
- `get isAsync():Boolean` – Informerar AIR-körningen om huruvida dataleverantörsobjektet tillhandahåller data asynkront eller synkront.
- `close():void` – Anropas av körningen när läsningen av data är helt slutförd (eller ett fel förhindrar fortsatt läsning). Du kan använda den här funktionen för att rensa resurser.
- `reportError( e:ErrorEvent ):void` – Anropas av körningen när ett fel inträffar vid läsningen av data.

Alla IFilePromise-metoder anropas av körningen under en dra och släpp-åtgärd där fillöftet används. Programlogiken skulle normalt inte anropa någon av dessa metoder direkt.

Använda en synkron dataleverantör i ett fillöfte

Adobe AIR 2 och senare

Det enklaste sättet att implementera gränssnittet IFilePromise är genom att använda ett synkront dataleverantörsobjekt, till exempel ett ByteArray eller ett synkront FileStream. I följande exempel skapas ett ByteArray-objekt, fylls med data och returneras när metoden `open()` anropas.

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
```

```
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

I praktiken är användbarheten för synkrona fillöften begränsad. Om mängden data är liten kan du lika enkelt skapa en fil i en tillfällig katalog och lägga till en vanlig fillistearray till Urklipp för dra och släpp. Det krävs däremot en lång synkron process om mängden data är stor, eller om det är resurskrävande att skapa dessa data. Långa synkrona processer kan blockera uppdateringar av användargränssnittet under märkbart lång tid och få det att verka som att programmet inte svarar. Om du vill undvika det här problemet kan du skapa en asynkron dataleverantör som styrs av en timer.

Använda en asynkron dataleverantör i ett fillöfte

Adobe AIR 2 och senare

När du använder ett asynkront dataleverantörsobjekt måste IFilePromise-egenskapen `isAsync` vara `true` och objektet som returneras av metoden `open()` måste implementera gränssnittet IEventDispatcher. Körningen lyssnar efter flera alternativa händelser så att olika inbyggda objekt kan användas som dataleverantör. Till exempel skickas `progress`-händelser av FileStream- och URLStream-objekt, medan `socketData`-händelser skickas av Socket-objekt. Körningen lyssnar efter lämpliga händelser från alla dessa objekt.

Följande händelser styr processen att läsa data från dataleverantörsobjektet:

- Event.OPEN – Informerar körningen om att datakällan är färdig.
- ProgressEvent.PROGRESS – Informerar körningen om att det finns tillgängliga data. Körningen kommer att läsa mängden befintliga data från dataleverantörsobjektet.
- ProgressEvent.SOCKET_DATA – Informerar körningen om att det finns tillgängliga data. Händelsen `socketData` skickas av socket-baserade objekt. För andra objekttyper bör du skicka en `progress`-händelse. (Körningen lyssnar efter båda objekten för att avgöra vilka data som kan läsas.)
- Event.COMPLETE – Informerar körningen om att alla data har lästs.
- Event.CLOSE – Informerar körningen om att alla data har lästs. (Körningen lyssnar av den här anledningen efter både `close` och `complete`.)
- IOErrorEvent.IOERROR – Informerar körningen om att ett fel uppstod när data lästes. Körningen avbryter skapandet av filen och anropar IFilePromise-metoden `close()`.

- SecurityErrorEvent.SECURITY_ERROR – Informerar körningen om att ett säkerhetsfel har inträffat. Körningen avbryter skapandet av filen och anropar IFilePromise-metoden `close()`.
- HTTPStatusEvent.HTTP_STATUS – Används av körningen tillsammans med `httpResponseStatus` för att säkerställa att tillgängliga data representerar önskat innehåll, och inte något felmeddelande (till exempel en 404-sida). Objekt som utgår från HTTP-protokollet bör skicka den här händelsen.
- HTTPStatusEvent.HTTP_RESPONSE_STATUS – Används av körningen tillsammans med `httpStatus` för att säkerställa att tillgängliga data representerar önskat innehåll. Objekt som utgår från HTTP-protokollet bör skicka den här händelsen.

Dataleverantören bör skicka de här händelserna i följande sekvens:

1. `open`-händelse
2. `progress`- eller `socketData`-händelser
3. `complete`- eller `close`-händelse

***Obs!** De inbyggda objekten FileStream, Socket, och URLStream skickar automatiskt de rätta händelserna.*

I följande exempel skapas ett fillöfte med hjälp av en anpassad, asynkron dataleverantör. Dataleverantörsklassen utökar ByteArray-objektet (för IDataInput-stöd) och implementerar gränssnittet IEventDispatcher. Vid varje timerhändelse genererar objektet en mängd data och skickar en förloppshändelse för att informera körningen om att dessa data finns tillgängliga. När en tillräckligt stor mängd data har producerats skickar objektet en slutförandehändelse.

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }
```

```
    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable,
bytesAvailable + chunkSize);
            dispatchEvent( progress )
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function,
useCapture:Boolean=false, priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function,
useCapture:Boolean=false):void
```

```
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

Obs! *Eftersom klassen AsyncDataProvider i exemplet utökar ByteArray, kan den inte också utöka EventDispatcher. Klassen använder ett internt EventDispatcher-objekt för att implementera gränssnittet IEventDispatcher och vidarebefordrar IEventDispatcher-metodanropen till det interna objektet. Du bör också utöka EventDispatcher och implementera IDataInput (eller implementera båda gränssnitten).*

Den asynkrona implementeringen av IFilePromise är nästan identisk med den synkrona implementeringen. De viktigaste skillnaderna är att `isAsync` returnerar `true` och att metoden `open()` returnerar ett asynkront dataobjekt:

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }
```

```
        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

Kapitel 36: Arbeta med menyer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd klasserna i API för snabbmenyer för att ändra snabbmenyer i webbaserade Flex- och Flash Player-program.

Använd klasserna i API:t för inbyggda menyer för att definiera program-, fönster, snabb- och popup-menyer i Adobe® AIR®.

Grunderna i menyer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande snabbstartartiklar på Adobe Developer Connection innehåller information med korta förklaringar och kodexempel på hur du skapar inbyggda menyer i AIR-program:

- Lägga till inbyggda menyer i ett AIR-program (Flex)
- Lägga till inbyggda menyer i ett AIR-program (Flash)

Med hjälp av klasserna för inbyggda menyer får du åtkomst till funktionerna för inbyggda menyer i det operativsystem som programmet körs på. NativeMenu-objekt kan användas för programmenyer (tillgängligt på Mac OS X), fönstermenyer (tillgängligt på Windows och Linux), kontextmenyer och popupmenyer.

Utanför AIR kan du använda snabbmenyklasserna för att ändra den snabbmeny som visas automatiskt i Flash Player när användaren högerklickar eller Kommando-klickar på ett objekt i programmet. (Det visas ingen automatisk snabbmeny för AIR-program.)

Menyklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Menyklasserna omfattar:

Paket	Klasser
flash.display	• NativeMenu • NativeMenuItem
flash.ui	• ContextMenu • ContextMenuItem
flash.events	• Event • ContextMenuEvent

Menyurval

Flash Player 9 och senare, Adobe AIR 1.0 och senare

AIR kan användas med följande typer av menyer:

Snabbmenyer Snabbmenyer öppnas när någon högerklickar eller Kommando-klickar på ett interaktivt objekt i SWF-innehåll eller på ett dokumentelement i HTML-innehåll.

I Flash Player-miljön visas en snabbmeny automatiskt. Du kan använda klasserna ContextMenu och ContextMenuItem för att lägga till egna kommandon på menyn. Du kan också ta bort vissa, men inte alla, av de inbyggda kommandona.

I AIR-miljön kan du skapa en snabbmeny med hjälp av klassen NativeMenu eller ContextMenu. I HTML-innehåll i AIR kan du använda WebKits HTML- och JavaScript-API:er för att lägga till snabbmenyer i HTML-element.

Programmenyer (endast AIR) En programmeny är en global meny som används i hela programmet. Programmenyer stöds på Mac OS X men inte på Windows eller Linux. På Mac OS X skapar operativsystemet automatiskt en programmeny. Du kan använda programmeringsgränssnittet för menyer i AIR för att lägga till alternativ och undermenyer till standardmenyerna. Du kan lägga till avlyssnare för hantering av befintliga menykommandon. Du kan även ta bort befintliga alternativ.

Fönstermenyer (endast AIR) En fönstermeny är kopplad till ett enda fönster och visas under namnlisten. Du kan lägga till menyer i ett fönster genom att skapa ett NativeMenu-objekt och tilldela det till `menu`-egenskapen i NativeWindow-objektet. Windows-menyer stöds på operativsystemen Windows och Linux men inte på Mac OS X. Inbyggda fönstermenyer kan bara användas med fönster som har systemfärg.

Menyer för dockan och aktivitetsfältet (endast AIR) Dessa ikonmenyer liknar kontextmenyer och tilldelas en programikon i dockan på Mac OS X eller i meddelandeområdet i aktivitetsfältet på Windows och Linux. Menyer för docknings- och systemfältsikoner använder klassen NativeMenu. På Mac OS X läggs alternativen till på menyn ovanför operativsystemets standardalternativ. På Windows och Linux finns ingen standardmeny.

Popup-menyer (endast AIR) Popup-menyerna i AIR fungerar som snabbmenyer men är inte alltid kopplade till särskilda programobjekt eller programkomponenter. Popup-menyer kan visas var som helst i ett fönster genom anrop av metoden `display()` i ett NativeMenu-objekt.

Anpassade menyer Inbyggda menyer ritas helt och hållet av operativsystemet och finns därför utanför Flash- och HTML-återgivningsmodellerna. I stället för att använda inbyggda menyer kan du alltid skapa egna, anpassade menyer med MXML, ActionScript eller JavaScript (i AIR). Sådana menyer måste återges helt i programinnehållet.

Flex-menyer Ramverket Adobe® Flex™ tillhandahåller en mängd Flex-menykomponenter. Flex-menyerna ritas av körningsmiljön i stället för operativsystemet och är inte *inbyggda* menyer. En Flex-menykomponent kan användas i Flex-fönster utan systemfärg. En annan fördel med Flex-menykomponenterna är att du kan ange menyerna deklarativt i MXML-format. Om du använder Flex-ramverket använder du Flex-menyklasserna för fönstermenyer i stället för inbyggda klasser.

Standardmenyer (endast AIR)

Följande standardmenyer finns i operativsystemet eller en inbyggd AIR-klass:

- Programmeny på Mac OS X
- Dockningsikonmeny på Mac OS X
- Snabbmeny för markerad text och markerade bilder i HTML-innehåll
- Snabbmeny för markerad text i ett TextField-objekt (eller ett objekt som utökar TextField)

Om snabbmenyer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I SWF-innehåll kan ett objekt som ärvs från InteractiveObject få en viss snabbmeny genom att ett menyobjekt tilldelas dess `contextMenu`-egenskap. Vissa kommandon finns med som standard, till exempel Framåt, Bakåt, Skriv ut, Kvalitet och Zoom. I AIR-miljön kan det menyobjekt som tilldelats `contextMenu` vara antingen av typen NativeMenu eller av typen ContextMenu. I Flash Player-miljön är bara klassen ContextMenu tillgänglig.

Du kan avlyssna händelser för inbyggda menyer eller snabbmenyer när du använder ContextMenu- och ContextMenuItem-klasserna, båda skickas. En fördel med ContextMenuEvent-objektegenskaperna är att contextMenuOwner identifierar det objekt som menyn hör till medan `mouseTarget` identifierar det objekt som användaren klickade på för att öppna menyn. Den här informationen är inte tillgänglig från NativeMenuEvent-objektet.

I följande exempel skapas ett Sprite-objekt och en enkel snabbmeny för redigering läggs till:

```
 var sprite:Sprite = new Sprite();
sprite.contextMenu = createContextMenu()
private function createContextMenu():ContextMenu{
    var editContextMenu:ContextMenu = new ContextMenu();
    var cutItem:ContextMenuItem = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItem = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItem = new ContextMenuItem("Paste")
    pasteItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doPasteCommand);
    editContextMenu.customItems.push(pasteItem);

    return editContextMenu
}
private function doCutCommand(event:ContextMenuEvent):void{trace("cut");}
private function doCopyCommand(event:ContextMenuEvent):void{trace("copy");}
private function doPasteCommand(event:ContextMenuEvent):void{trace("paste");}
```

Obs! *Till skillnad från SWF-innehåll som visas i en webbläsarmiljö har inte snabbmenyer i AIR några inbyggda kommandon.*

Anpassa en snabbmeny i Flash Player

I en webbläsare eller projektor har snabbmenyer i SWF-innehåll alltid inbyggda objekt. Du kan ta bort alla dessa standardkommandon från menyn, utom kommandona Inställningar och Om. Genom att ställa in scenegenskapen `showDefaultContextMenu` till `false` tas dessa kommandon bort från snabbmenyn.

Om du vill skapa en egen snabbmeny för ett speciellt visningsobjekt skapar du en ny instans av klassen ContextMenu, anropar metoden `hideBuiltInItems()` och tilldelar instansen egenskapen `contextMenu` för den DisplayObject-instansen. I nästa exempel finns en dynamiskt ritad fyrkant med ett snabbmenykommando som ändrar dess färg slumpmässigt:

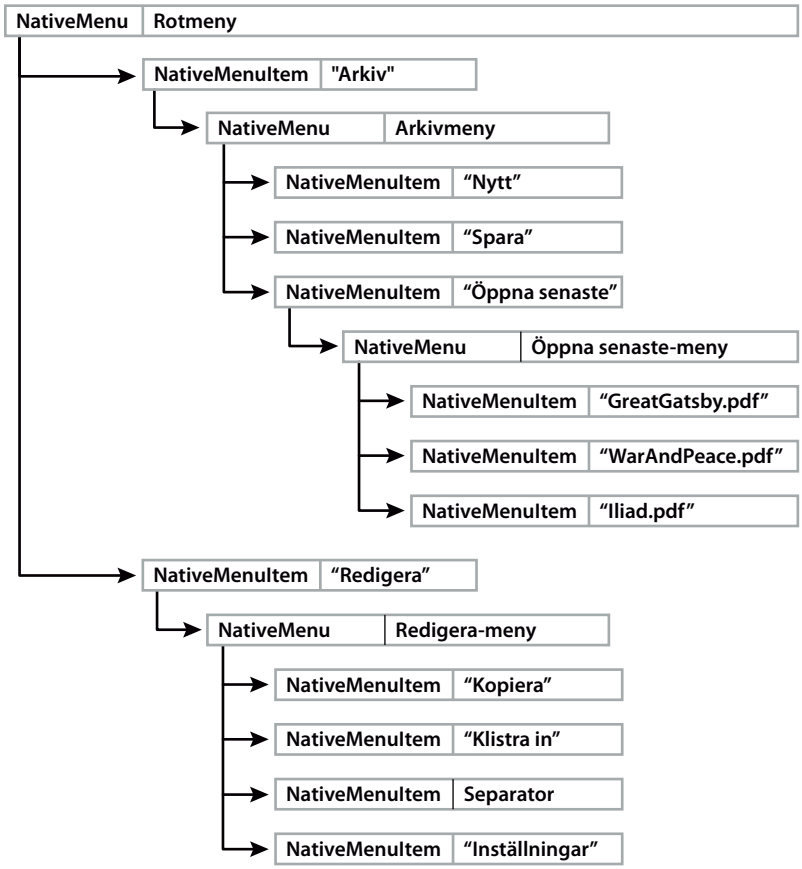
```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

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);
}
```

Struktur för inbyggda menyer (AIR)

Adobe AIR 1.0 och senare

Inbyggda menyer är hierarkiska. NativeMenu-objekt innehåller underordnade NativeMenuItem-objekt. NativeMenuItem-objekt som representerar undermenyer kan i sin tur innehålla NativeMenu-objekt. Menyobjekten på topp- och rotnivå i strukturen representerar menyraden för program- och fönstermenyer. (I snabbmenyer, ikonmenyer och popup-menyer finns ingen menyrad.)

Följande diagram visar strukturen för en vanlig meny. Rotmenyn representerar menyrader och innehåller två menyalternativ som refererar till en *File*-undermeny och en *Edit*-undermeny. File-undermenyn i den här strukturen innehåller två kommandoalternativ och ett alternativ som refererar till en *Open Recent Menu*-undermeny, som själv innehåller tre alternativ. Edit-undermenyn innehåller tre kommandon och en avgränsare.



Om du vill definiera en undermeny krävs både ett NativeMenu-objekt och ett NativeMenuItem-objekt. NativeMenuItem-objektet definierar den etikett som visas på den överordnade menyn och gör att användaren kan öppna undermenyn. NativeMenu-objektet fungerar som en behållare för alternativ på undermenyn. NativeMenuItem-objektet refererar till NativeMenu-objektet via `submenu`-egenskapen för NativeMenuItem.

Ett exempel med kod som skapar den här menyn finns i ”Exempel på inbyggd meny: Fönster- och programmeny (AIR)” på sidan 646.

Menyhändelser

Adobe AIR 1.0 och senare

NativeMenu- och NativeMenuItem-objekt skickar båda `preparing`-, `displaying`- och `select`-händelser:

Preparing: När objektet är på väg att inleda en användarinteraktion skickar menyn och dess menyalternativ en `preparing`-händelse till alla registrerade avlyssnare. Till interaktioner räknas även att öppna menyn eller välja ett alternativ med ett kortkommando.

Obs! *`preparing`-händelsen är bara tillgänglig för Adobe AIR 2.6 och senare versioner.*

Displaying: Direkt innan en meny visas skickar menyn och dess menyalternativ en `displaying`-händelse till registrerade avlyssnare.

Med `preparing`- och `displaying`-händelserna kan du uppdatera menyinnehållet och alternativen innan allt visas för användaren. I avlyssnaren för `displaying`-händelsen i en ”Open Recent”-meny kan du till exempel ändra menyalternativen så att de återspeglar den aktuella listan över senast visade dokument.

Om du tar bort det menyalternativ vars kortkommando utlöste en `preparing`-händelse avbryts menyinteraktionen och ingen `select`-händelse skickas.

Egenskaperna `target` och `currentTarget` för händelsen är båda det objekt för vilket lyssnaren har registrerats: antingen själva menyn eller ett av dess alternativ.

`preparing`-händelsen skickas före `displaying`-händelsen. Du lyssnar vanligtvis efter den ena eller andra händelsen, inte båda.

Select: När ett kommandoalternativ väljs av användaren skickar alternativet en `select`-händelse till alla registrerade avlyssnare. Undermeny- och avgränsaralternativ kan inte väljas och kan därför aldrig skicka en `select`-händelse.

En `select`-händelse ”bubblar upp” från ett menyalternativ till dess övre meny och upp till rotmenyn. Du kan lyssna efter `select`-händelser direkt vid ett alternativ och du kan lyssna högre upp i menystrukturen. När du lyssnar efter `select`-händelsen på en meny kan du identifiera det valda alternativet med hjälp av händelsens `target`-egenskap. När händelsen ”bubblar upp” genom menyhierarkin identifierar händelseobjektets `currentTarget`-egenskap det aktuella menyobjektet.

Obs! *ContextMenu- och ContextMenuItem-objekt skickar `menuItemSelect`- och `menuSelect`-händelser liksom `select`-, `preparing`- och `displaying`-händelser.*

Tangentkombinationer för inbyggda menykommandon (AIR)

Adobe AIR 1.0 och senare

Du kan tilldela en tangentkombination (kallas ibland kortkommando) till ett menykommando. Menyalternativen skickar en `select`-händelse till alla registrerade avlyssnare när tangenten eller tangentkombinationen trycks ned. Menyn som innehåller alternativet måste ingå i programmets meny eller det aktiva fönstret för att kommandot ska kunna anropas.

Tangentkombinationerna består av två delar – en sträng som representerar den primära tangenten och en array med modifieringstangenter som också måste tryckas ned. Om du vill tilldela en primär tangent anger du menyalternativets `keyEquivalent`-egenskap som teckensträngen för den tangenten. Om du använder en versal läggs skifttangenten automatiskt till i modifieringsarrayen.

På Mac OS X är kommandotangenten standardmodifierare (`Keyboard.COMMAND`). På Windows och Linux är det CTRL-tangenten (`Keyboard.CONTROL`). De här standardtangenterna läggs automatiskt till i modifieringsarrayen. Om du vill tilldela andra modifieringstangenter tilldelar du en ny array, som innehåller de önskade tangentkoderna, till egenskapen `keyEquivalentModifiers`. Standardarrayen skrivs över. Oavsett om du använder standardmodifierare eller tilldelar en egen modifieringsarray, så läggs skifttangenten till om den sträng du tilldelar egenskapen `keyEquivalent` är en versal. Konstanterna för de tangentkoder som ska användas för modifieringstangenterna definieras i klassen Keyboard.

Strängen för den tilldelade tangentkombinationen visas automatiskt bredvid menyalternativnamnet. Formatet beror på användarens operativsystem och systeminställningar.

Obs! *Om du tilldelar värdet* `Keyboard.COMMAND` *till en tangentmodifieringsarray i operativsystemet Windows visas ingen tangentkombination på menyn. Ctrl-tangenten måste dock användas för att aktivera menykommandot.*

I följande exempel tilldelas `Ctrl+Skift+G` som tangentkombination för ett menyalternativ:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

I det här exemplet tilldelas `Ctrl+Skift+G` som tangentkombination genom att ange modifieringsarrayen direkt:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [Keyboard.CONTROL];
```

Obs! *Tangentkombinationer aktiveras bara för program- och fönstermenyer. Om du lägger till en tangentkombination till en snabbmeny eller popup-meny visas tangentkombinationen på menyetiketten, men det tillhörande menykommandot anropas aldrig.*

Kortkommandon (AIR)

Adobe AIR 1.0 och senare

Kortkommandon är en del av operativsystemets tangentbordsgränssnitt för menyer. På Linux, Mac OS X och Windows kan användare öppna menyer och välja kommandon med tangentbordet, men det finns vissa mindre skillnader.

På Mac OS X skriver användaren de första bokstäverna i meny- eller kommandonamnet och trycker sedan på returtangenten. Egenskapen `mnemonicIndex` ignoreras.

I Windows är det bara en enda bokstav som behövs. Som standard är den signifikanta bokstaven det första tecknet i etiketten, men om du tilldelar ett kortkommando till menyalternativet så blir det signifikanta tecknet den tilldelade bokstaven. Om två alternativ på en meny har samma signifikanta tecken (oavsett om ett kortkommando har tilldelats eller inte) ändras användarens tangentbordsåtgärder för menyn en aning. I stället för att trycka på en enda bokstav för att välja menyn eller kommandot måste användaren trycka på bokstaven så många gånger som behövs för att markera önskat alternativ och sedan trycka på Retur-tangenten för att avsluta valet. För att upprätthålla ett konsekvent beteende bör du tilldela varje objekt i en fönstermeny ett unikt kortkommandotecken.

På Linux finns inga förvalda kortkommandotecken. Du måste ange ett värde för egenskapen `mnemonicIndex` för ett menyobjekt för att tilldela det ett kortkommandotecken.

Ange kortkommandotecknet som ett index i etikettsträngen. Index för det första tecknet i en etikett är 0. Om du vill använda ”r” som kortkommando för ett menyalternativ med etiketten ”Format” ska du ange du ett värde för egenskapen `mnemonicIndex` som är lika med 2.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

Lägen för menyalternativ

Adobe AIR 1.0 och senare

Menyalternativ har två lägesegenskaper, `checked` och `enabled`:

checked Ange som `true` om du vill visa en kryssmarkering bredvid alternativetiketten.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

enabled Växla mellan värdena `true` och `false` om du vill bestämma vilket kommando som är aktiverat. Inaktiverade alternativ är gråtonade och skickar inga `select`-händelser.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

Koppla ett objekt till ett menyalternativ

Adobe AIR 1.0 och senare

Med hjälp av klassen NativeMenuItems `data`-egenskap kan du referera till ett godtyckligt objekt i varje alternativ. På en ”Open Recent”-meny kan du till exempel tilldela File-objektet för varje dokument till varje menyalternativ.

```
var file:File = File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

Skapa inbyggda menyer (AIR)

Adobe AIR 1.0 och senare

I det här avsnittet beskriver vi hur du skapar olika sorters inbyggda menyer som AIR kan hantera.

Skapa ett rotmenyobjekt

Adobe AIR 1.0 och senare

Om du vill skapa ett NativeMenu-objekt som fungerar som rot för menyn använder du konstruktorn NativeMenu:

```
var root:NativeMenu = new NativeMenu();
```

För program- och fönstermenyer representerar rotmenyn menyraden och ska bara innehålla alternativ som öppnar undermenyer. Snabbmenyer och popup-menyer har ingen menyrad; rotmenyn kan därför innehålla både kommandon, avgränsare och undermenyer.

När menyn har skapats kan du lägga till menyalternativ. Menyalternativen visas i den ordning som de lades till, om du inte lägger till alternativen i ett särskilt index med hjälp av metoden `addItemAt()` i ett menyobjekt.

Tilldela menyn som en program-, fönster-, ikon eller snabbmeny eller visa den som en popup-meny som i följande avsnitt:

Ställa in programmenyn eller fönster-menyn

Det är viktigt att koden omfattar både programmenyer (stöds på Mac OS) och fönstermenyer (stöds på andra operativsystem)

```
var root:NativeMenu = new NativeMenu();
if (NativeApplication.supportsMenu)
{
    NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

Obs! *Mac OS definierar en meny som innehåller standardalternativ för varje program. Om du tilldelar ett nytt NativeMenu-objekt till* `menu`*-egenskapen i NativeApplication-objektet ersätts standardmenyn. Du kan också använda standardmenyn i stället för att ersätta den.*

I Adobe Flex finns en FlexNativeMenu-klass för att enkelt skapa menyer som fungerar på olika plattformar. Om du använder Flex Framework kan du använda klasserna FlexNativeMenu i stället för klassen NativeMenu.

Ställa in en snabbmeny i ett interaktivt objekt

```
interactiveObject.contextMenu = root;
```

Ställa in dockningsikonens meny eller systemfältsikonens meny

Det är viktigt att koden omfattar både programmenyer (stöds på Mac OS) och fönstermenyer (stöds på andra operativsystem)

```
if (NativeApplication.supportsSystemTrayIcon)
{
    SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
}
else if (NativeApplication.supportsDockIcon)
{
    DockIcon(NativeApplication.nativeApplication.icon).menu = root;
}
```

Obs! *Mac OS X definierar en standardmeny för programmets dockningsikon. Om du tilldelar ett nytt NativeMenu-objekt till menu-egenskapen i DockIcon-objektet, visas alternativen på den menyn ovanför standardalternativen. Du kan inte ta bort, få tillgång till eller ändra standardmenyalternativen.*

Visa en meny som en popup-meny

```
root.display(stage, x, y);
```

Fler hjälpavsnitt

Utveckla AIR-program som fungerar på flera plattformar

Skapa en undermeny

Adobe AIR 1.0 och senare

Om du vill skapa en undermeny lägger du till ett NativeMenuItem-objekt till den överordnade menyn och tilldelar sedan det NativeMenu-objekt som definierar undermenyn till alternativets `submenu`-egenskap. I AIR finns två sätt att skapa undermenyalternativ och deras tillhörande menyobjekt:

Du kan skapa ett menyalternativ och dess tillhörande menyobjekt i ett steg med hjälp av metoden `addSubmenu()`:

```
var editMenuItem:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

Du kan också skapa menyalternativet och tilldela menyobjektet till dess `submenu`-egenskap separat:

```
var editMenuItem:NativeMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new NativeMenu();
```

Skapa ett menykommando

Adobe AIR 1.0 och senare

Om du vill skapa ett menykommando lägger du till ett NativeMenuItem-objekt till en meny och lägger till en händelseavlyssnare som refererar den funktion som implementerar menykommandot:

```
var copy:NativeMenuItem = new NativeMenuItem("Copy", false);
copy.addEventListener(Event.SELECT, onCopyCommand);
editMenu.addItem(copy);
```

Du kan lyssna efter `select`-händelsen i själva kommandoalternativet (visas i exemplet) eller också kan du lyssna efter `select`-händelsen i ett överordnat menyobjekt.

Obs! *Menyalternativ som representerar undermenyer och avgränsare skickar inte `select`-händelser och kan därför inte användas som kommandon.*

Skapa en menyavgränsningslinje

Adobe AIR 1.0 och senare

Om du vill skapa en avgränsningslinje skapar du ett NativeMenuItem-objekt genom att ange `isSeparator`-parametern i konstruktorn som `true`. Lägg sedan till avgränsningsalternativet på rätt plats i menyn:

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
editMenu.addItem(separatorA);
```

Den etikett som angetts för avgränsaren (om sådan finns) visas inte.

Om snabbmenyer i HTML (AIR)

Adobe AIR 1.0 och senare

I HTML-innehåll som visas med objektet HTMLLoader kan händelsen `contextmenu` användas för att visa en snabbmeny. En snabbmeny visas som standard automatiskt när användaren anropar snabbmenyhändelsen för markerad text (genom att högerklicka eller kommandoklicka på texten). Om du vill förhindra att standardmenyn öppnas lyssnar du efter `contextmenu`-händelsen och anropar händelseobjektets `preventDefault()`-metod:

```
function showContextMenu(event){
    event.preventDefault();
}
```

Du kan sedan visa en anpassad snabbmeny med hjälp av DHTML-tekniker eller genom att visa en inbyggd AIR-snabbmeny. I följande exempel visas en inbyggd snabbmeny genom anrop av menyns `display()`-metod som svar på `contextmenu`-händelsen i HTML:

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context
menu.</p>
</body>
</html>
```

Visa inbygga popup-menyer (AIR)

Adobe AIR 1.0 och senare

Du kan visa ett NativeMenu-objekt på valfri tid och plats ovanför ett fönster genom att anropa menyns `display()`-metod. Den här metoden kräver en referens till scenen. Därför kan bara innehåll i programsandlådan visa en meny som en popup-meny.

Följande metod visar den meny som definieras av ett NativeMenu-objekt med namnet `popupMenu` och som är svar på en musklickning:

```
private function onMouseClick(event:MouseEvent):void {
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

Obs! *Menyn måste inte visas som direkt svar på en händelse. Alla metoder kan anropa funktionen* `display()`.

Hantera menyhändelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En meny skickar händelser när användaren väljer menyn eller när användaren väljer ett menyalternativ.

Sammanfattning av händelser för menyklasser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Lägg till händelseavlyssnare till menyer eller enskilda alternativ för att hantera menyhändelser.

Object	Skickade händelser
NativeMenu (AIR)	Event.PREPARING (Adobe AIR 2.6 och senare) Event.DISPLAYING Event.SELECT (spridda från underordnade objekt och undermenyer)
NativeMenuItem (AIR)	Event.PREPARING (Adobe AIR 2.6 och senare) Event.SELECT Event.DISPLAYING (spridda från överordnade menyer)
ContextMenu	ContextMenuEvent.MENU_SELECT
ContextMenuItem	ContextMenuEvent.MENU_ITEM_SELECT Event.SELECT (AIR)

Välja menyhändelser

Adobe AIR 1.0 och senare

Om du vill hantera en klickning på ett menyalternativ lägger du till en händelseavlyssnare för `select`-händelsen i NativeMenuItem-objektet:

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommandX.addEventListener(Event.SELECT, doCommandX)
```

Eftersom `select`-händelser ”bubblar upp” till menyerna ovanför kan du även lyssna efter select-händelser på en överordnad meny. När du lyssnar på en menynivå kan du använda händelseobjektets `target`-egenskap för att avgöra vilket menykommando som valdes. I följande exempel spåras etiketten för det valda kommandot:

```
 var colorMenuItem:NativeMenuItem = new NativeMenuItem("Choose a color");
var colorMenu:NativeMenu = new NativeMenu();
colorMenuItem.submenu = colorMenu;

var red:NativeMenuItem = new NativeMenuItem("Red");
var green:NativeMenuItem = new NativeMenuItem("Green");
var blue:NativeMenuItem = new NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(NativeApplication.supportsMenu){
    NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT, colorChoice);
} else if (NativeWindow.supportsMenu){
    var windowMenu:NativeMenu = new NativeMenu();
    this.stage.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(Event.SELECT, colorChoice);
}

function colorChoice(event:Event):void {
    var menuItem:NativeMenuItem = event.target as NativeMenuItem;
    trace(menuItem.label + " has been selected");
}
```

Om du använder klassen ContextMenuItem kan du lyssna efter `select`-händelsen eller `menuItemSelect`-händelsen. `menuItemSelect`-händelsen ger dig ytterligare information om det objekt som har snabbmenyn, men den ”bubblar” inte upp till menyerna ovanför.

Visa menyhändelser

Adobe AIR 1.0 och senare

Om du vill hantera en menys öppnande kan du lägga till en lyssnare för `displaying`-händelsen som skickas innan en meny visas. Du kan använda visningshändelsen för att uppdatera menyn, t.ex. genom att lägga till eller ta bort alternativ eller genom att uppdatera aktiverade eller markerade statusar för enskilda alternativ. Du kan också lyssna efter händelsen `menuSelect` från ett ContextMenu-objekt.

I AIR 2.6 och senare versioner kan du använda `preparing`-händelsen för att uppdatera en meny som svar på antingen att en meny visas eller att ett alternativ väljs via ett kortkommando.

Exempel på inbyggd meny: Fönster- och programmeny (AIR)

Adobe AIR 1.0 och senare

I följande exempel skapas den meny som visas i ”Struktur för inbyggda menyer (AIR)” på sidan 637.

Menyn har utformats så att den fungerar med både Windows (som bara stöder fönstermenyer) och Mac OS X (som bara stöder programmenyer). För att avgöra skillnaden mellan dem kontrollerar klassen MenuExamples konstruktor de statiska `supportsMenu`-egenskaperna i klasserna NativeWindow och NativeApplication. Om `NativeWindow.supportsMenu` är `true` skapar konstruktorn ett NativeMenu-objekt för fönstret, och skapar sedan och lägger till undermenyerna File och Edit. Om `NativeApplication.supportsMenu` är `true` skapar och lägger konstruktorn till menyerna File och Edit till den befintliga menyn som finns i Mac OS X-operativsystemet.

Exemplet visar också menyhändelsehantering. `select`-händelsen hanteras på alternativnivån och på menynivån. Varje meny i kedjan, från den meny som innehåller det valda alternativet till rotmenyn, svarar på `select`-händelsen. `displaying`-händelsen används med ”Open Recent”-menyn. Alldeles innan menyn öppnas uppdateras alternativen på menyn från den senaste Documents-arrayen (ändras inte i det här exemplet). Även om det inte visas i det här exemplet kan du faktiskt lyssna efter `displaying`-händelser i enskilda alternativ.

```
package {
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filesystem.File;
    import flash.desktop.NativeApplication;

    public class MenuExample extends Sprite
    {
        private var recentDocuments:Array =
            new Array(new File("app-storage:/GreatGatsby.pdf"),
                      new File("app-storage:/WarAndPeace.pdf"),
                      new File("app-storage:/Iliad.pdf"));

        public function MenuExample()
        {
            var fileMenu:NativeMenuItem;
            var editMenu:NativeMenuItem;

            if (NativeWindow.supportsMenu){
                stage.nativeWindow.menu = new NativeMenu();
                stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);
                fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }

            if (NativeApplication.supportsMenu){
                NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT,
selectCommandMenu);
                fileMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }
        }

        public function createFileMenu():NativeMenu {
```

```
        var fileMenu:NativeMenu = new NativeMenu();
        fileMenu.addEventListener(Event.SELECT, selectCommandMenu);

        var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New"));
        newCommand.addEventListener(Event.SELECT, selectCommand);
        var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save"));
        saveCommand.addEventListener(Event.SELECT, selectCommand);
        var openRecentMenu:NativeMenuItem =
                fileMenu.addItem(new NativeMenuItem("Open Recent"));
        openRecentMenu.submenu = new NativeMenu();
        openRecentMenu.submenu.addEventListener(Event.DISPLAYING,
                                        updateRecentDocumentMenu);
        openRecentMenu.submenu.addEventListener(Event.SELECT, selectCommandMenu);

        return fileMenu;
    }

    public function createEditMenu():NativeMenu {
        var editMenu:NativeMenu = new NativeMenu();
        editMenu.addEventListener(Event.SELECT, selectCommandMenu);

        var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy"));
        copyCommand.addEventListener(Event.SELECT, selectCommand);
        copyCommand.keyEquivalent = "c";
        var pasteCommand:NativeMenuItem =
                editMenu.addItem(new NativeMenuItem("Paste"));
        pasteCommand.addEventListener(Event.SELECT, selectCommand);
        pasteCommand.keyEquivalent = "v";
        editMenu.addItem(new NativeMenuItem("", true));
        var preferencesCommand:NativeMenuItem =
                editMenu.addItem(new NativeMenuItem("Preferences"));
        preferencesCommand.addEventListener(Event.SELECT, selectCommand);

        return editMenu;
    }

    private function updateRecentDocumentMenu(event:Event):void {
        trace("Updating recent document menu.");
        var docMenu:NativeMenu = NativeMenu(event.target);

        for each (var item:NativeMenuItem in docMenu.items) {
            docMenu.removeItem(item);
        }

        for each (var file:File in recentDocuments) {
            var menuItem:NativeMenuItem =
                    docMenu.addItem(new NativeMenuItem(file.name));
            menuItem.data = file;
            menuItem.addEventListener(Event.SELECT, selectRecentDocument);
        }
    }

    private function selectRecentDocument(event:Event):void {
        trace("Selected recent document: " + event.target.data.name);
    }

    private function selectCommand(event:Event):void {
```

```
            trace("Selected command: " + event.target.label);
        }

        private function selectCommandMenu(event:Event):void {
            if (event.currentTarget.parent != null) {
                var menuItem:NativeMenuItem =
                        findItemForMenu(NativeMenu(event.currentTarget));
                if (menuItem != null) {
                    trace("Select event for \"" +
                            event.target.label +
                            "\" command handled by menu: " +
                            menuItem.label);
                }
            } else {
                trace("Select event for \"" +
                        event.target.label +
                        "\" command handled by root menu.");
            }
        }

        private function findItemForMenu(menu:NativeMenu):NativeMenuItem {
            for each (var item:NativeMenuItem in menu.parent.items) {
                if (item != null) {
                    if (item.submenu == menu) {
                        return item;
                    }
                }
            }
            return null;
        }
    }
}
```

Kapitel 37: Aktivitetsfältsikoner i AIR

Adobe AIR 1.0 och senare

I många operativsystem finns ett aktivitetsfält (till exempel Dock i Mac OS X) som kan innehålla en ikon som representerar ett program. I Adobe® AIR® finns ett gränssnitt för att interagera med programikoner i aktivitetsfältet via egenskapen `NativeApplication.nativeApplication.icon`.

- Använda docknings- och systemfältsikoner (Flex)
- Använda ikoner i systemfältet och Dock (Flash)

Fler hjälpavsnitt

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

Om aktivitetsfältsikoner

Adobe AIR 1.0 och senare

I AIR skapas `NativeApplication.nativeApplication.icon`-objektet automatiskt. Objekttypen är antingen DockIcon eller SystemTrayIcon, beroende på operativsystemet. Du kan ta reda på vilka av dessa InteractiveIcon-underklasser som AIR stöder på det aktuella operativsystemet med hjälp av egenskaperna `NativeApplication.supportsDockIcon` och `NativeApplication.supportsSystemTrayIcon`. InteractiveIcon-basklassen innehåller egenskaperna `width`, `height` och `bitmaps`. Med dessa egenskaper kan du ändra bilden som används för ikonen. Om du använder egenskaper som är specifika för DockIcon eller SystemTrayIcon på fel operativsystem genereras dock ett körningsfel.

Om du vill välja en bild eller ändra bilden som används för en ikon skapar du en array som innehåller en eller flera bilder och tilldelar den till egenskapen `NativeApplication.nativeApplication.icon.bitmaps`. Storleken på aktivitetsfältsikonerna kan variera mellan olika operativsystem. Du kan undvika att bildkvaliteten försämras på grund av skalning genom att lägga till flera storlekar på bilderna i `bitmaps`-arrayen. Om du lägger till fler än en bild väljs automatiskt den storlek som bäst matchar aktivitetsfältsikonens aktuella visningsstorlek, och bilden skalas vid behov. I följande exempel anges bilden för en aktivitetsfältsikon med två bilder:

```
NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

Om du vill ändra ikonbilden tilldelar du en array som innehåller den nya bilden (eller bilderna) till egenskapen `bitmaps`. Du kan animera ikonen genom att ändra bilden som svar på en `enterFrame`- eller `timer`-händelse.

Om du vill ta bort ikonen från meddelandeområdet i Windows och Linux, eller återställa ikonernas standardutseende i Mac OS X, tilldelar du en tom array till `bitmaps`:

```
NativeApplication.nativeApplication.icon.bitmaps = [];
```

Dockningsikoner

Adobe AIR 1.0 och senare

AIR stöder dockningsikoner när `NativeApplication.supportsDockIcon` har värdet `true`. Egenskapen `NativeApplication.nativeApplication.icon` representerar programikonen i Dock (inte en fönsterikon).

Obs! *AIR stöder inte ändring av fönsterikoner i Dock i Mac OS X. Dessutom gäller endast ändringar i programikonerna i Dock när ett program körs. Ikonen återfår sitt normala utseende när programmet avslutas.*

Ikonmenyer i Dock

Adobe AIR 1.0 och senare

Du kan lägga till kommandon på standardmenyn i Dock genom att skapa ett NativeMenu-objekt som innehåller kommandona och tilldela det till egenskapen `NativeApplication.nativeApplication.icon.menu`. Objekten på menyn visas ovanför alternativen på standardmenyn i Dock.

Studsa dockan

Adobe AIR 1.0 och senare

Du kan studsa dockningsikonen genom att anropa metoden `NativeApplication.nativeApplication.icon.bounce()`. Om du anger Informational för parametern `bounce()` `priority` studsar ikonen en gång. Om du anger Critical studsar ikonen tills användaren aktiverar programmet. Konstanter för parametern `priority` definieras i klassen NotificationType.

Obs! *Ikonen studsar inte om programmet redan är aktivt.*

Dockningsikonhändelser

Adobe AIR 1.0 och senare

När du klickar på dockningsikonen skickar NativeApplication-objektet en `invoke`-händelse. Om programmet inte är igång startas det automatiskt. I annat fall skickas `invoke`-händelsen till den programinstans som körs.

Systemfältsikoner

Adobe AIR 1.0 och senare

AIR stöder systemfältsikoner när `NativeApplication.supportsSystemTrayIcon` är `true`, vilket för närvarande är fallet endast på Windows och de flesta Linux-distributioner. På Windows och Linux visas systemfältsikoner i aktivitetsfältets meddelandeområde. Ingen ikon visas som standard. Om du vill visa en ikon tilldelar du en array som innehåller BitmapData-objekt till egenskapen `bitmaps` för ikonen. Om du vill ändra ikonbilden tilldelar du en array som innehåller de nya bilderna till `bitmaps`. Om du vill ta bort ikonen anger du värdet `null` för `bitmaps`.

Ikonmenyer i systemfältet

Adobe AIR 1.0 och senare

Du kan lägga till en meny till systemfältsikonen genom att skapa ett NativeMenu-objekt och tilldela det till egenskapen `NativeApplication.nativeApplication.icon.menu` (ingen standardmeny tillhandahålls i operativsystemet). Du öppnar systemfältsikonens meny genom att högerklicka på ikonen.

Knappbeskrivningar för systemfältsikoner

Adobe AIR 1.0 och senare

Du lägger till en knappbeskrivning för en ikon genom att ange egenskapen tooltip:

```
NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

Händelser för systemfältsikoner

Adobe AIR 1.0 och senare

Det SystemTrayIcon-objekt som egenskapen NativeApplication.nativeApplication.icon refererar till skickar en ScreenMouseEvent för `click`-, `mouseDown`-, `mouseUp`-, `rightClick`-, `rightMouseDown`- och `rightMouseUp`-händelser. Du kan använda dessa händelser tillsammans med en ikonmeny så att användarna kan interagera med ditt program när det saknar synliga fönster.

Exempel: Skapa ett program utan fönster

Adobe AIR 1.0 och senare

I följande exempel skapas ett AIR-program som har en systemfältsikon, men inga synliga fönster. (Egenskapen `visible` för programmet kan inte vara angivet till `true` i programbeskrivningen eftersom fönstret då visas när programmet startas.)

```
package
{
    import flash.display.Loader;
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.desktop.DockIcon;
    import flash.desktop.SystemTrayIcon;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.desktop.NativeApplication;

    public class SysTrayApp extends Sprite
    {
        public function SysTrayApp():void{
            NativeApplication.nativeApplication.autoExit = false;
            var icon:Loader = new Loader();
            var iconMenu:NativeMenu = new NativeMenu();
            var exitCommand:NativeMenuItem = iconMenu.addItem(new NativeMenuItem("Exit"));
                exitCommand.addEventListener(Event.SELECT, function(event:Event):void {
```

```
                NativeApplication.nativeApplication.icon.bitmaps = [];
                NativeApplication.nativeApplication.exit();
            });

        if (NativeApplication.supportsSystemTrayIcon) {
            NativeApplication.nativeApplication.autoExit = false;
            icon.contentLoaderInfo.addEventListener(Event.COMPLETE, iconLoadComplete);
            icon.load(new URLRequest("icons/AIRApp_16.png"));

            var systray:SystemTrayIcon =
                NativeApplication.nativeApplication.icon as SystemTrayIcon;
            systray.tooltip = "AIR application";
            systray.menu = iconMenu;
        }

        if (NativeApplication.supportsDockIcon){
            icon.contentLoaderInfo.addEventListener(Event.COMPLETE,iconLoadComplete);
            icon.load(new URLRequest("icons/AIRApp_128.png"));
            var dock:DockIcon = NativeApplication.nativeApplication.icon as DockIcon;
            dock.menu = iconMenu;
        }
    }

    private function iconLoadComplete(event:Event):void
    {
        NativeApplication.nativeApplication.icon.bitmaps =
            [event.target.content.bitmapData];
    }
  }
}
```

Obs! *När du använder Flex WindowedApplication-komponenten måste du ange attributet* `visible` *för WindowedApplication-taggen till* `false`*. Detta attribut åsidosätter inställningen i programbeskrivningen.*

Obs! *I exemplet antas att programkatalogen innehåller bildfiler med namnen* `AIRApp_16.png` *och* `AIRApp_128.png` *i underkatalogen* `icons`*. (I AIR SDK finns exempel på ikonfiler som du kan kopiera till projektmappen.)*

Ikoner och knappar i Aktivitetsfältet i Windows

Adobe AIR 1.0 och senare

Ikoner som representerar fönster visas vanligtvis i fönsterområdet i Aktivitetsfältet eller i Dock så att användarna enkelt kan få åtkomst till bakgrunder eller minimerade fönster. I Dock i Mac OS X visas både en ikon för ditt program och en ikon för varje minimerat fönster. Aktivitetsfälten i Microsoft Windows och Linux visar en knapp med programikonen och titeln för varje fönster av normaltyp i ditt program.

Markera fönsterknappen i Aktivitetsfältet

Adobe AIR 1.0 och senare

När ett fönster är i bakgrunden kan du meddela användaren att det har inträffat en händelse som rör fönstret. I Mac OS X kan du meddela användaren genom att studsa programikonen i Dock (se "Studsa dockan" på sidan 651). På Windows and Linux kan du markera knappen på aktivitetsfältet genom att anropa `notifyUser()`-metoden i NativeWindow-instansen. Parametern `type` som skickas till metoden anger hur viktigt meddelandet är:

- `NotificationType.CRITICAL`: fönsterikonen blinkar tills användaren placerar fönstret i förgrunden.
- `NotificationType.INFORMATIONAL`: fönsterikonen markeras genom att den ändrar färg.

  Obs! *På Linux stöds endast meddelanden av informativ typ. Om du överför ett typvärde till funktionen `notifyUser()` får du samma resultat.*

  Följande programsats markerar fönsterknappen i Aktivitetsfältet:

```
stage.nativeWindow.notifyUser(NotificationType.CRITICAL);
```

  Det händer ingenting om metoden `NativeWindow.notifyUser()` anropas i ett operativsystem som saknar stöd för fönstermeddelanden. Använd egenskapen `NativeWindow.supportsNotification` om du vill ta reda på om fönstermeddelanden stöds.

Skapa fönster utan knappar eller ikoner i Aktivitetsfältet

Adobe AIR 1.0 och senare

I operativsystemet Windows visas inte fönster som har skapats med typerna *utility* eller *lightweight* i Aktivitetsfältet. Osynliga fönster visas inte heller i Aktivitetsfältet.

Eftersom det ursprungliga fönstret har typen *normal* måste du antingen stänga det ursprungliga fönstret eller låta det vara osynligt för att kunna skapa ett program utan att några fönster visas i Aktivitetsfältet. Om du vill stänga alla fönster i programmet utan att avsluta programmet, anger du värdet `false` för NativeApplication-objektets `autoExit`-egenskap innan du stänger det sista fönstret. Om du vill förhindra att det ursprungliga fönstret överhuvudtaget visas, lägger du till `<visible>false</visible>` för programbeskrivningsfilens `<initalWindow>`-element (och försäkra dig om att inte `visible`-egenskapen har värdet `true` eller att metoden `activate()` för fönstret anropas).

I nya fönster som öppnas av programmet anger du `NativeWindowType.UTILITY` eller `NativeWindowType.LIGHTWEIGHT` för `type`-egenskapen för det NativeWindowInitOption-objekt som skickas till fönsterkonstruktorn.

I Mac OS X visas minimerade fönster i Dock. Du kan förhindra att den minimerade ikonen visas genom att dölja fönstret i stället för att minimera det. I följande exempel görs en lyssning efter en `nativeWindowDisplayState`-ändringshändelse, och händelsen avbryts om fönstret minimeras. I stället anges värdet `false` för fönstrets `visible`-egenskap med hanteraren:

```
private function preventMinimize(event:NativeWindowDisplayStateEvent):void{
    if(event.afterDisplayState == NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

Om ett fönster är minimerat i Dock i Mac OS X när du anger värdet `false` för egenskapen `visible` tas dockningsikonen inte bort. Användarna kan fortfarande visa fönstret igen genom att klicka på ikonen.

Kapitel 38: Arbeta med filsystemet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash® Player har grundläggande funktioner för att läsa och skriva filer via klassen FileReference. Av säkerhetsskäl måste användaren vid körning i Flash Player alltid ge tillåtelse innan du kan läsa eller skriva en fil.

Adobe® AIR® ger mer fullständig tillgång till värddatorns filsystem än Flash Player. Med hjälp av filsystems-API:t i AIR får du tillgång till och kan hantera kataloger och filer, du kan skapa kataloger och filer, skriva data till filer och så vidare.

Fler hjälpavsnitt

flash.net.FileReference

flash.net.FileReferenceList

flash.filesystem.File

flash.filesystem.FileStream

Använda klassen FileReference

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett FileReference-objekt representerar en datafil på en klient- eller serverdator. Med metoderna för klassen FileReference kan du låta programmet läsa in och spara datafiler lokalt, och överföra fildata till och från fjärrservrar.

Med klassen FileReference kan du läsa in, överföra och spara datafiler på två olika sätt. Klassen FileReference har sedan den introducerades innehållit metoderna `browse()`, `upload()` och `download()`. Använd metoden `browse()` om du vill låta användaren välja en fil. Använd metoden `upload()` om du vill överföra filens data till en fjärrserver. Använd metoden `download()` om du vill hämta data från servern och spara dem i en lokal fil. Från och med Flash Player 10 och Adobe AIR 1.5 innehåller klassen FileReference även metoderna `load()` och `save()`. Med metoderna `load()` och `save()` kan du även få åtkomst till och spara lokala filer direkt. Användningen av dessa metoder påminner om metoderna med samma namn i klasserna URLLoader och Loader.

Obs! *Klassen File, som utökar klassen FileReference, och klassen FileStream innehåller ytterligare funktioner för att arbeta med filer och det lokala filsystemet. Klasserna File och FileStream stöds bara i AIR, inte i Flash Player.*

Klassen FileReference

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Varje FileReference-objekt representerar en enskild datafil på den lokala datorn. Egenskaperna för klassen FileReference innehåller information om filens storlek, typ, namn, filnamnstillägg, datum när den skapades och ändringsdatum.

Obs! *Egenskapen `creator` stöds bara på Mac OS. Alla övriga plattformar returnerar `null`.*

Obs! *Egenskapen `extension` stöds endast i Adobe AIR.*

Du kan skapa en instans av klassen FileReference på två olika sätt:

- Använda operatorn `new` så som visas i följande kod:

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- Anropa metoden `FileReferenceList.browse()` som öppnar en dialogruta där användaren ombeds välja en eller flera filer som ska överföras. Sedan skapas en array med FileReference-objekt om användaren väljer en eller flera filer.

När du har skapat ett FileReference-objekt kan du göra följande:

- Anropa metoden `FileReference.browse()` som öppnar en dialogruta där användaren ombeds välja en enda fil från det lokala filsystemet. Detta sker vanligtvis före ett efterföljande anrop till metoden `FileReference.upload()` eller `FileReference.load()`. Anropa metoden `FileReference.upload()` om du vill skicka filen till en fjärrserver. Anropa metoden `FileReference.load()` om du vill öppna en lokal fil.
- Anropa metoden `FileReference.download()`. Metoden `download` öppnar en dialogruta, där användaren kan välja den plats där en ny fil sparas. Sedan överförs data från servern och de lagras i den nya filen.
- Anropa metoden `FileReference.load()`. Den här metoden börjar läsa in data från en fil som tidigare valts med metoden `browse()`. Metoden `load()` kan inte anropas förrän `browse()`-åtgärden slutförts (användaren väljer en fil).
- Anropa metoden `FileReference.save()`. Den här metoden öppnar en dialogruta och uppmanar användaren att välja en filplats i det lokala filsystemet. Metoden sparar sedan data på den angivna platsen.

Obs! *Du kan bara köra en `browse()`-, `download()`- eller `save()`-åtgärd åt gången eftersom bara en dialogruta kan vara öppen i taget.*

FileReference-objektegenskaperna, till exempel `name`, `size` och `modificationDate`, definieras inte förrän något av följande inträffar:

- Metoden `FileReference.browse()` eller metoden `FileReferenceList.browse()` har anropats, och användaren har valt en fil i dialogrutan.
- Metoden `FileReference.download()` har anropats, och användaren har angett en plats för en ny fil i dialogrutan.

Obs! *Vid nedladdning fylls bara egenskapen `FileReference.name` i innan nedladdningen är klar. När filen har laddats ned är alla egenskaper tillgängliga.*

När anropen till metoden `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` eller `FileReference.save()` körs fortsätter uppspelningen av SWF-filen i de flesta spelare, och händelser skickas och kod körs.

Vid både överföring och nedladdning kan en SWF-fil bara komma åt filer i sin egen domän, inklusive de domäner som anges med en principfil. Du måste lägga upp en principfil på servern som innehåller filen om servern inte finns i samma domän som SWF-filen som initierar överföringen eller hämtningen.

Se även FileReference.

Läsa in data från filer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `FileReference.load()` kan du läsa in data från en lokal fil till minnet.

Obs! *Koden måste först anropa metoden `FileReference.browse()` så att användaren kan välja vilken fil som ska läsas in. Den här begränsningen gäller inte innehåll som körs i Adobe AIR i programmets säkerhetssandlåda*

Metoden `FileReference.load()` returneras omedelbart efter att den anropats, men informationen som läses in är inte tillgänglig direkt. FileReference-objektet skickar händelser för att anropa avlyssnarmetoder vid varje steg i inläsningsprocessen.

FileReference-objektet skickar följande händelser under inläsningen.

- `open` event (`Event.OPEN`): Skickas när inläsningen startar.
- `progress` event (`ProgressEvent.PROGRESS`): Skickas regelbundet allt eftersom byte med data läses in från filen.
- `complete` event (`Event.COMPLETE`): Skickas när inläsningsåtgärden har slutförts.
- `ioError` event (`IOErrorEvent.IO_ERROR`): Skickas om inläsningen misslyckas på grund av ett I/O-fel som inträffar när filen öppnas eller när data läses från filen.

När FileReference-objektet skickar hela händelsen blir den inlästa informationen tillgänglig som en ByteArray i FileReference-objektets `data`egenskap.

I följande exempel visas hur du uppmanar användaren att välja en fil och sedan läsa in data från den filen i minnet:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
```

```
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

Exempelkoden skapar först FileReference-objektet `fileRef` och anropar sedan objektets `browse()`-metod. Metoden `browse()` öppnar en dialogruta, som uppmanar användaren att välja en fil. När en fil har valts anropar koden metoden `onFileSelected()`. Den här metoden lägger till lyssnare för händelserna `progress` och `complete` och anropar sedan FileReference-objektets `load()`-metod. De andra hanterarmetoderna i exemplet returnerar bara meddelanden om inläsningens förlopp. När inläsningen är klar visas innehållet i den inlästa filen med hjälp av metoden `trace()`.

I Adobe AIR erbjuder klassen FileStream ytterligare funktioner för att läsa data från en lokal fil. Du hittar mer information i ”Läsa och skriva filer” på sidan 691.

Spara data i lokala filer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `FileReference.save()` kan du spara data i en lokal fil. Den börjar med att öppna en dialogruta där användaren kan ange ett nytt filnamn och en plats där filen ska sparas. När användaren har valt filnamn och plats sparas data i den nya filen. När filen har sparats fylls egenskaperna i FileReference-objektet i med egenskaperna för den lokala filen.

Obs! *Koden kan bara anropa metoden `FileReference.save()` som svar på en användarinitierad händelse, till exempel en musklicks- eller tangentbordshändelse. Annars returneras ett fel. Den här begränsningen gäller inte innehåll som körs i Adobe AIR i programmets säkerhetssandlåda.*

Metoden `FileReference.save()` returneras omedelbart efter att den anropats. FileReference-objektet skickar sedan händelser för att anropa avlyssnarmetoder vid varje steg medan filen sparas.

FileReference-objektet skickar följande händelser medan filen sparas.

- `select` event (`Event.SELECT`): Skickas när användaren anger plats och filnamn för den nya filen som ska sparas.
- `cancel` event (`Event.CANCEL`): Skickas när användaren klickar på knappen Avbryt i dialogrutan.
- `open` event (`Event.OPEN`): Skickas när filen har sparats.
- `progress` event (`ProgressEvent.PROGRESS`): Skickas regelbundet allt eftersom byte med data sparas i filen.
- `complete` event (`Event.COMPLETE`): Skickas när spara-åtgärden har slutförts.

- `ioError` event (`IOErrorEvent.IO_ERROR`): Skickas om det inte går att spara på grund av ett I/O-fel som inträffar när data ska sparas i filen.

Den typ av objekt som skickas i `data`-parametern för metoden `FileReference.save()` avgör hur informationen skrivs till filen:

- Om det är ett String-värde sparas informationen som en textfil med UTF-8-kodning.
- Om det är ett XML-objekt skrivs informationen till en fil i XML-format med all formatering bevarad.
- Om det är ett ByteArray-objekt skrivs innehållet direkt till filen utan konvertering.
- Om det är någon annan typ av objekt anropar metoden `FileReference.save()` objektets `toString()`-metod och sparar sedan det resulterande String-värdet i en UTF-8-textfil. Om objektets `toString()`-metod inte kan anropas returneras ett fel.

Om värdet på `data`-parametern är `null` returneras ett fel.

Följande kod bygger på exemplet som beskrivs ovan för metoden `FileReference.load()`. När data har lästs från filen uppmanar exempelkoden användaren att ange ett filnamn och sparar sedan data i en ny fil:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
```

```
        }
        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.save(fileRef.data,"NewFileName.txt");
        }

        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

När alla data har lästs in från filen anropar koden metoden `onComplete()`. Metoden `onComplete()` tar bort lyssnarna för inläsningshändelserna och anropar sedan metoden `saveFile()`. Metoden `saveFile()` anropar metoden `FileReference.save()`. Metoden `FileReference.save()` öppnar en ny dialogruta där användaren kan ange ett nytt filnamn och en plats där filen ska sparas. Resten av händelseavlyssnarmetoderna spårar förloppet när filen sparas tills detta är klart.

I Adobe AIR erbjuder klassen FileStream ytterligare funktioner för att skriva data till en lokal fil. Du hittar mer information i ”Läsa och skriva filer” på sidan 691.

Överföra filer till en server

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill överföra filer till en server anropar du först metoden `browse()` så att användaren kan välja en eller flera filer. Sedan, när metoden `FileReference.upload()` anropas, överförs den valda filen till servern. Om användaren väljer flera filer med hjälp av metoden `FileReferenceList.browse()` skapar Flash Player en array med de valda filerna vid namn `FileReferenceList.fileList`. Du kan sedan använda metoden `FileReference.upload()` för att överföra varje fil för sig.

Obs! *Metoden `FileReference.browse()` gör att du bara kan överföra enskilda filer. Om du vill tillåta att användare kan överföra flera filer använder du metoden `FileReferenceList.browse()`.*

Som standard kan användare välja valfri filtyp på den lokala datorn i dialogrutan för att välja fil. Utvecklare kan ange en eller flera anpassade filtypsfilter genom klassen FileFilter och skicka en array av filterinstanser för filer till metoden `browse()`:

```
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);
```

När användaren har valt filer och klickat på knappen Öppna i systemets filväljare, skickas händelsen `Event.SELECT`. Om metoden `FileReference.browse()` används för att välja filen som ska överföras skickas filen till en webbserver med följande kod:

```
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");
}
```

Du kan skicka data till servern med metoden `FileReference.upload()` genom att använda egenskaperna `URLRequest.method` och `URLRequest.data` för att skicka variabler med hjälp av metoden `POST` eller `GET`.

När du försöker överföra en fil med hjälp av metoden `FileReference.upload()` skickas följande händelser:

- `open` event (`Event.OPEN`): Skickas när överföringen startar.
- `progress` event (`ProgressEvent.PROGRESS`): Skickas regelbundet allt eftersom byte med data överförs.
- `complete` event (`Event.COMPLETE`): Skickas när överföringen har slutförts.
- `httpStatus` event (`HTTPStatusEvent.HTTP_STATUS`): Skickas när överföringen misslyckas på grund av ett HTTP-fel.
- `httpResponseStatus` event (`HTTPStatusEvent.HTTP_RESPONSE_STATUS`): Skickas om ett anrop till metoden `upload()` eller `uploadUnencoded()` försöker få åtkomst till data via HTTP, och Adobe AIR upptäcker och returnerar statuskoden för begäran.
- `securityError`-händelse (`SecurityErrorEvent.SECURITY_ERROR`): Skickas om en överföring misslyckas på grund av en säkerhetsöverträdelse.
- `uploadCompleteData` event (`DataEvent.UPLOAD_COMPLETE_DATA`): Skickas när data har tagits emot från servern efter en slutförd överföring.
- `ioError` event (`IOErrorEvent.IO_ERROR`): Skickas om överföringen misslyckas på grund av någon av följande orsaker:
  - Ett in-/utdatafel inträffade när Flash Player läser, skriver eller skickar filen.
  - SWF-filen försökte att överföra en fil till en server som kräver verifiering (till exempel användarnamn och lösenord). Under överföring kan inte användaren ange lösenord i Flash Player.

- Parametern `url` innehåller ett ogiltigt protokoll. Metoden `FileReference.upload()` måste använda antingen HTTP eller HTTPS.

Flash Player erbjuder inte fullständigt stöd för servrar som kräver verifiering. Bara SWF-filer som körs i en webbläsare med hjälp av webbläsarplugin-programmet eller Microsoft ActiveX®-kontrollen kan erbjuda en dialogruta där användaren uppmanas att ange användarnamn och lösenord för verifiering, och då bara för nedladdning. För överföring med plugin-programmet eller med ActiveX-kontrollen och för överföring/nedladdning med en fristående eller extern spelare misslyckas filöverföringen.

Om du vill skapa ett serverskript i ColdFusion för att godta en filöverföring från Flash Player kan du använda kod som följande:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Den här ColdFusion-koden överför filen som har skickats av Flash Player och sparar den i samma katalog som ColdFusion-mallen. Om katalogen redan innehåller en fil med samma namn, skrivs den filen över. Ovanstående kod visar den minsta mängd kod som krävs för att godta en filöverföring. Skriptet bör inte användas i verkliga produktionsmiljöer. Bäst är att lägga till datavalidering för att se till att användare endast överför giltiga filtyper, till exempel en bild i stället för ett potentiellt farligt serverskript.

Koden nedan visar filöverföringar med hjälp av PHP, och inkluderar datavalidering. Skriptet begränsar antalet överförda filer i överföringskatalogen till 10, vilket garanterar att filen är mindre än 200 kB, och tillåter bara att JPEG-, GIF- och PNG-filer överförs till och sparas i filsystemet.

```
<?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;
    }
}
?>
```

Du kan skicka ytterligare variabler till överföringsskriptet med hjälp av förfrågningsmetoden `POST` eller `GET`. Om du vill skicka ytterligare `POST`-variabler till överföringsskriptet kan du använda följande kod:

```
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");
}
```

Ovanstående exempel skapar ett URLVariables-objekt som du skickar till fjärrskriptet på servern. I tidigare versioner av ActionScript kunde du skicka variabler till överföringsskriptet på servern genom att skicka värden i frågesträngen. I ActionScript 3.0 kan du skicka variabler till fjärrskriptet med hjälp av ett URLRequest-objekt, som låter dig skicka data med metoden `POST` eller `GET`, vilket i sin tur gör det möjligt att enkelt och smidigt skicka större mängder data. För att kunna ange om variablerna skickas med förfrågningsmetoden `GET` eller `POST` kan du ställa in egenskapen `URLRequest.method` på `URLRequestMethod.GET` respektive `URLRequestMethod.POST`.

I ActionScript 3.0 kan du även åsidosätta standardfältnamnet på överföringsfilen, `Filedata`, genom att ange en andra parameter för metoden `upload()`, som i föregående exempel (som ersatte standardvärdet `Filedata` med `Custom1`).

Flash Player gör som standard inte någon testöverföring, men du kan kringgå detta genom att skicka värdet `true` som en tredje parameter till metoden `upload()`. Syftet med testöverföringen är att kontrollera att den riktiga filöverföringen och serververifieringen (om någon) kommer att lyckas.

Obs! *Testöverföring går än så länge bara att göra på Windows-baserade Flash Player.*

I serverskriptet som hanterar filöverföringen bör en HTTP `POST`-begäran ingå med följande element:

- `Content-Type` med värdet `multipart/form-data`.
- `Content-Disposition` med ett `name`-attribut inställt på `Filedata` och ett `filename`-attribut inställt på originalfilens namn. Du kan ange ett anpassat `name`-attribut genom att skicka ett värde för parametern `uploadDataFieldName` i metoden `FileReference.upload()`.
- Filens binära innehåll

Här följer ett exempel på en HTTP `POST`-begäran:

```
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
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

Följande exempel på HTTP `POST`-begäran skickar tre `POST`-variabler: `api_sig`, `api_key` och `auth_token`, och använder ett anpassat namnvärde för överföringsdatafält (`photo`):

```
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

------------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"

XXXXXXXXXXXXXXXXXXXXXXXX
------------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--
```

Laddar ned filer från en server

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan låta användare ladda ned filer från en server med hjälp av metoden `FileReference.download()`, som tar två parametrar: `request` och `defaultFileName`. Den första parametern är URLRequest-objektet som innehåller webbadressen för filen som ska laddas ned. Den andra parametern är valfri och gör att du kan ange ett standardnamn som visas i nedladdningsdialogrutan. Om du utelämnar den andra parametern, `defaultFileName`, används filnamnet från den angivna webbadressen.

Följande kod hämtar en fil med namnet index.xml från samma katalog som SWF-filen:

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Om du vill ställa in standardnamnet till currentnews.xml i stället för index.xml anger du parametern `defaultFileName` som i följande utdrag:

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Det kan vara bra att ändra namnet på filen om serverfilnamnet inte är intuitivt eller om det genererades av servern. Det är även bra att uttryckligen ange parametern `defaultFileName` när du laddar ned en fil med hjälp av ett skript på servern i stället för att ladda ned filen direkt. Du måste till exempel ange parametern `defaultFileName` om du har ett skript på servern som laddar ned specifika filer baserat på vilka URL-variabler som skickas till det. Annars är standardnamnet på den nedladdade filen detsamma som namnet på skriptet på servern.

Data kan skickas till servern med hjälp av metoden `download()` genom att bifoga parametrar till webbadressen som serverskriptet ska analysera. Följande ActionScript 3.0-utdrag laddar ned ett dokument baserat på vilka parametrar som skickas till ett ColdFusion-skript:

```
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.");
            }
        }
    }
}
```

Följande kod visar ColdFusion-skriptet, download.cfm, som laddar ned en av två filer från servern beroende på en URL-variabels värde:

```
<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>
```

Klassen FileReferenceList

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen FileReferenceList gör att användaren kan välja en eller flera filer som ska överföras till ett skript på servern. Filöverföringen hanteras av metoden `FileReference.upload()`, som måste anropas för varje fil som användaren väljer.

Följande kod skapar två FileFilter-objekt (`imageFilter` och `textFilter`) och skickar dem i en array till metoden `FileReferenceList.browse()`. Detta gör att operativsystemets fildialogruta innehåller två möjliga filter för filtyper.

```
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.");
}
```

Att låta användaren välja och överföra en eller flera filer med hjälp av klassen FileReferenceList är detsamma som att använda `FileReference.browse()` för att välja filer, även om FileReferenceList låter dig välja flera filer. Vid överföring av flera filer måste du överföra varje fil med hjälp av `FileReference.upload()`, som i följande kod:

```
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");
}
```

Eftersom händelsen `Event.COMPLETE` läggs in i varje enskilt FileReference-objekt i arrayen, anropar Flash Player metoden `completeHandler()` när varje enskild fil har överförts.

Använda filsystems-API:t i AIR

Adobe AIR 1.0 och senare

Filsystems-API:t i Adobe AIR innehåller följande klasser:

- Fil
- FileMode
- FileStream

Du kan göra följande (och mer) med filsystems-API:t:

- Kopiera, skapa, ta bort och flytta filer och kataloger
- Hämta information om filer och kataloger
- Läsa och skriva filer

Grunderna för AIR-filer

Adobe AIR 1.0 och senare

Följande snabbstartartiklar på Adobe Developer Connection innehåller information med korta förklaringar och kodexempel på hur du arbetar med filsystemet i AIR:

- Skapa en textfilsredigerare (Flash)
- Skapa en textfilsredigerare (Flex)
- Skapa ett program för katalogsökning (Flex)
- Reading and writing from an XML preferences file (Läsa och skriva från en XML-inställningsfil) (Flex)
- Komprimera filer och data (Flex)

I Adobe AIR finns klasser som du kan använda när du vill få tillgång till, skapa och hantera både filer och mappar. Dessa klasser, som finns i paketet flash.filesystem, används så här:

File-klasser	Beskrivning
File	Ett File-objekt som representerar en sökväg till en fil eller katalog. Du använder ett filobjekt när du vill skapa en pekare till en fil eller mapp, och börja arbeta med den filen eller mappen.
FileMode	Klassen FileMode definierar strängkonstanter som används i `fileMode`-parametern i metoderna `open()` och `openAsync()` i klassen FileStream. `fileMode`-parametern för de här metoderna avgör vilka funktioner som är tillgängliga för FileStream-objektet när filen väl öppnas, som omfattar att skriva, läsa, lägga till och uppdatera.
FileStream	FileStream-objekt används för att öppna filer som ska läsas och skrivas. När du har skapat ett File-objekt som pekar på en ny eller befintlig fil, skickar du den pekaren till FileStream-objektet så att du kan öppna den och skriva eller läsa data.

En del metoder i klassen File har både synkrona och asynkrona versioner:

- `File.copyTo()` och `File.copyToAsync()`

- `File.deleteDirectory()` och `File.deleteDirectoryAsync()`
- `File.deleteFile()` och `File.deleteFileAsync()`
- `File.getDirectoryListing()` och `File.getDirectoryListingAsync()`
- `File.moveTo()` och `File.moveToAsync()`
- `File.moveToTrash()` och `File.moveToTrashAsync()`

FileStream-åtgärder fungerar dessutom synkront eller asynkront beroende på hur FileStream-objektet öppnar filen: genom att anropa metoden `open()` eller anropa metoden `openAsync()`.

Med de asynkrona versionerna kan du starta processer som körs i bakgrunden och skicka händelser när de är klara (eller när felhändelser uppstår). Annan kod kan köras medan dessa asynkrona bakgrundsprocesser utförs. När det gäller asynkrona versioner av åtgärderna måste du ställa in händelseavlyssnarfunktioner, med metoden `addEventListener()` för File- eller FileStream-objektet som anropar funktionen.

Med de synkrona versionerna kan du skriva enklare kod där du inte behöver ställa in händelseavlyssnare. Eftersom annan kod inte kan köras när en synkron metod körs, kan det hända att viktiga processer som återgivning och animering av visningsobjekt fördröjs.

Arbeta med File-objekt i AIR

Adobe AIR 1.0 och senare

Ett File-objekt är en pekare till en fil eller en katalog i filsystemet.

Klassen File utökar klassen FileReference. Klassen FileReference, som finns i både Adobe® Flash® Player och AIR, utgör en pekare till en fil, men klassen File lägger till egenskaper och metoder som av säkerhetsskäl inte visas i Flash Player (i en SWF-fil som körs i en webbläsare).

Om klassen File

Adobe AIR 1.0 och senare

Du kan använda klassen File för följande:

- Hämta sökvägen till specialkataloger, bland annat användarkatalogen, användarens dokumentkatalog, katalogen som programmet startades från samt programkatalogen.
- Kopiera filer och kataloger.
- Flytta filer och kataloger.
- Ta bort filer och kataloger (eller flytta dem till papperskorgen)
- Visa filer och kataloger som finns i en katalog
- Skapa tillfälliga filer och mappar

När ett File-objekt pekar på en filsökväg, kan du använda den om du vill läsa och skriva fildata, med hjälp av klassen FileStream.

Ett File-objekt kan peka på sökvägen till en fil eller katalog som inte längre finns. Du kan använda sådana File-filobjekt när du skapar en fil eller katalog.

Sökvägar till File-objekt

Adobe AIR 1.0 och senare

Varje File-objekt har två egenskaper som definierar sökvägen:

Egenskap	Beskrivning
`nativePath`	Anger den plattformsspecifika sökvägen till en fil. I Windows kan en sökväg vara "c:\Sample directory\test.txt" medan den i Mac OS är "/Sample directory/test.txt". `nativePath`-egenskapen använder ett omvänt snedstreck (\) som katalogavgränsare i Windows, och ett vanligt snedstreck (/) i Mac OS och Linux.
`url`	Denna kan använda filens URL-schema för att peka mot en fil. I Windows kan en sökväg vara "file:///c:/Sample%20directory/test.txt" medan den i Mac OS är "file:///Sample%20directory/test.txt". Körningsversionen omfattar andra särskilda URL-scheman förutom `file` och dessa beskrivs i "URL-scheman som stöds i AIR" på sidan 679

Klassen File inkluderar statiska egenskaper som du använder för att peka på standardkataloger i Mac OS, Windows och Linux. Bland dessa egenskaper finns:

- `File.applicationStorageDirectory` – en katalog som är unik för varje installerat AIR-program. Den här katalogen är ett bra ställe att lagra dynamiska programresurser och användarinställningar. Stora datamängder bör lagras någon annanstans.

  På Android och iOS tas programkatalogen bort när programmet avinstalleras eller när användaren väljer att rensa programdata, men detta gäller inte andra plattformar.

- `File.applicationDirectory` – den katalog där programmet installeras (tillsammans med eventuella resurser). På vissa operativsystem lagras programmet i en enda paketfil, i stället för i en fysisk katalog. I så fall kan det hända att innehållet inte är tillgängligt via systemsökvägen. Programkatalogen är skrivskyddad.
- `File.desktopDirectory` – användarens skrivbordskatalog. Om plattformen inte definierar någon skrivbordskatalog används en annan plats i filsystemet.
- `File.documentsDirectory` – användarens dokumentkatalog. Om plattformen inte definierar någon dokumentkatalog används en annan plats i filsystemet.
- `File.userDirectory` – användarkatalogen. Om plattformen inte definierar någon användarkatalog används en annan plats i filsystemet.

Obs! *Om plattformen inte definierar några standardplatser för skrivbords-, dokument- eller användarkataloger kan `File.documentsDirectory`, `File.desktopDirectory` och `File.userDirectory` referera till samma katalog.*

Dessa egenskaper har olika värden på olika operativsystem. Mac och Windows har till exempel olika systemsökvägar till användarens skrivbordskatalog. Egenskapen `File.desktopDirectory` pekar däremot på en lämplig katalogsökväg på alla plattformar. Om du vill skriva program som fungerar bra på olika plattformar använder du de här egenskaperna som utgångspunkt för att referera till andra kataloger och filer som används av programmet. Sedan använder du metoden `resolvePath()` för att närmare specificera sökvägen. Den här koden pekar till exempel på filen preferences.xml i programmets katalog:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

Även om du kan använda klassen File för att peka på en viss sökväg kan det leda till att programmet inte fungerar på alla plattformar. Sökvägen C:\Documents and Settings\joe\ fungerar till exempel bara på Windows. Därför är det bäst att använda de statiska egenskaperna i klassen File, som `File.documentsDirectory`.

Vanliga katalogplatser

Plattform	Katalogtyp	Normal filsystemplats
Android	Program	`/data/data/`
	Programlagring	`/data/data/air.applicationID/filename/Local Store`
	Cache	/data/data/applicationID/cache
	Skrivbord	`/mnt/sdcard`
	Dokument	`/mnt/sdcard`
	Tillfällig	`/data/data/applicationID/cache/FlashTmp.randomString`
	Användare	`/mnt/sdcard`
iOS	Program	`/var/mobile/Applications/uid/filename.app`
	Programlagring	`/var/mobile/Applications/uid/Library/Application Support/applicationID/Local Store`
	Cache	/var/mobile/Applications/uid/Library/Caches
	Skrivbord	ej tillgängligt
	Dokument	`/var/mobile/Applications/uid/Documents`
	Tillfällig	`/private/var/mobile/Applications/uid/tmp/FlashTmpNNN`
	Användare	ej tillgängligt
Linux	Program	`/opt/filename/share`
	Programlagring	`/home/userName/.appdata/applicationID/Local Store`
	Skrivbord	`/home/userName/Desktop`
	Dokument	`/home/userName/Documents`
	Tillfällig	`/tmp/FlashTmp.randomString`
	Användare	`/home/userName`
Macintosh	Program	`/Applications/filename.app/Contents/Resources`
	Programlagring	`/Users/`***`userName`***`/Library/Preferences/`***`applicationid`***`/Local Store` (AIR 3.2 och tidigare) ***`path`***`/Library/Application Support/`***`applicationid`***`/Local Store` (AIR 3.3 och senare), där sökvägen är antingen `/Users/`***`userName`***`/Library/Containers/`***`bundle-id`***`/Data` (sandlådemiljö) eller `/Users/userName` (vid körning utanför sandlådemiljö)
	Cache	/Users/userName/Library/Caches
	Skrivbord	`/Users/`***`userName`***`/Desktop`
	Dokument	`/Users/`***`userName`***`/Documents`
	Tillfällig	`/private/var/folders/JY/randomString/TemporaryItems/FlashTmp`
	Användare	`/Users/`***`userName`***

Plattform	Katalogtyp	Normal filsystemplats
Windows	Program	`C:\Program Files\filename`
	Programlagring	`C:\Documents and settings\userName\ApplicationData\applicationID\Local Store`
	Cache	C:\Documents and settings\userName\Local Settings\Temp
	Skrivbord	`C:\Documents and settings\userName\Desktop`
	Dokument	`C:\Documents and settings\userName\My Documents`
	Tillfällig	`C:\Documents and settings\userName\Local Settings\Temp\randomString.tmp`
	Användare	`C:\Documents and settings\userName`

De faktiska systemsökvägarna för dessa kataloger varierar beroende på operativsystem och datorkonfiguration. Sökvägarna i den här tabellen är exempel. Du bör alltid använda lämpliga statiska egenskaper för klassen File för att referera till de här katalogerna så att programmet fungerar som det ska oavsett plattform. I ett riktigt AIR-program hämtas värdena för `applicationID` och `filename` (som visas i tabellen) från programbeskrivningsfilen. Om du anger ett utgivar-ID i programbeskrivningsfilen läggs utgivar-ID:t till program-ID:t i dessa sökvägar. Värdet på `userName` är kontonamnet för den användare som installerar programmet.

Peka ett File-objekt mot en katalog

Adobe AIR 1.0 och senare

Du kan ställa in ett File-objekt på två sätt så att det pekar mot en katalog.

Peka mot användarens hemkatalog

Adobe AIR 1.0 och senare

Du kan peka ett File-objekt mot användarens hemkatalog. Följande kod ställer in ett File-objekt att peka mot en AIR Test-underkatalog till hemkatalogen:

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

Peka mot användarens dokumentkatalog

Adobe AIR 1.0 och senare

Du kan peka ett File-objekt mot användarens dokumentkatalog. Följande kod ställer in ett File-objekt att peka mot en AIR Test-underkatalog till dokumentkatalogen:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test");
```

Peka mot skrivbordskatalogen

Adobe AIR 1.0 och senare

Du kan peka ett File-objekt mot skrivbordet. Följande kod ställer in ett File-objekt att peka mot en AIR Test-underkatalog till skrivbordet:

```
var file:File = File.desktopDirectory.resolvePath("AIR Test");
```

Peka mot programlagringskatalogen

Adobe AIR 1.0 och senare

Du kan peka ett File-objekt mot programlagringskatalogen. För alla AIR-program finns det en unik associerad sökväg som definierar programlagringskatalogen. Den här katalogen är unik för varje program och användare. Du kan använda den här katalogen för att lagra användarspecifika, programspecifika data (till exempel användardata eller inställningsfiler). Följande kod pekar ett File-objekt mot en inställningsfil, prefs.xml, som finns i programlagringskatalogen:

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

Sökvägen för programlagringskatalogen baseras vanligtvis på användarnamnet och program-ID:t. Följande filsystemplatser visas här för att underlätta felsökning av programmet. Du bör alltid använda egenskapen `File.applicationStorage` eller URI-schemat `app-storage:` för att lösa filkonflikter i den här katalogen:

- På Mac OS – varierar beroende på AIR-version:

  AIR 3.2 och tidigare: `/Users/användarnamn/Library/Preferences/`*`applicationID`*`/Local Store/`

  AIR 3.3 och senare: *`path`*`/Library/Application Support/`*`applicationID`*`/Local Store`, där *`path`* antingen är `/Users/`*`username`*`/Library/Containers/`*`bundle-id`*`/Data` (sandlådemiljö) eller `/Users/`*`username`* (vid körning utanför sandlådemiljö)

  Exempel (AIR 3.2):

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- I Windows (i katalogen Documents and Settings):

  C:\Documents and Settings\användarnamn`\Application Data\`*program-ID*`\Local Store\`

  Till exempel:

  ```
  C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
  ```

- I Linux – i:

  `/home/`*användarnamn*`/.appdata/`*program-ID*`/Local Store/`

  Till exempel:

  ```
  /home/babbage/.appdata/com.example.TestApp/Local Store
  ```

- I Android:

  /data/data/`androidPackageID`/`applicationID/Local Store`

  Till exempel:

  ```
  /data/data/air.com.example.TestApp/com.example.TestApp/Local Store
  ```

Obs! *Om ett program har ett utgivar-ID används även detta utgivar-ID som en del av sökvägen till programlagringskatalogen.*

URL:en (och egenskapen `url`) för ett File-objekt som har skapats med `File.applicationStorageDirectory` använder URL-schemat `app-storage` (se ”URL-scheman som stöds i AIR” på sidan 679), så här:

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
```

Peka mot programkatalogen

Adobe AIR 1.0 och senare

Du kan peka ett File-objekt mot katalogen där programmet installerades, som också kallas för programkatalogen. Du kan referera till den här katalogen med hjälp av egenskapen `File.applicationDirectory`. Du kan använda den här katalogen för att undersöka programbeskrivningsfilen eller andra resurser som har installerats med programmet. Följande kod pekar ett File-objekt mot en katalog med namnet *images* i programkatalogen:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
```

URL:en (och egenskapen `url`) för ett File-objekt som har skapats med `File.applicationDirectory` använder URL-schemat `app` (se ”URL-scheman som stöds i AIR” på sidan 679), så här:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

Obs! *I Android är filerna i programpaketet inte tillgängliga via `nativePath`. Egenskapen `nativePath` är en tom sträng. Använd alltid URL:en för att komma åt filer i programkatalogen i stället för en systemsökväg.*

Peka mot cachekatalogen

Adobe AIR 3.6 och senare

Du kan peka ett File-objekt mot operativsystemets tillfälliga katalog eller cachekatalog med egenskapen `File.cacheDirectory`. Den här katalogen innehåller tillfälliga filer som inte behövs för att programmet ska kunna köras, och användaren råkar inte ut för problem eller dataförlust om de tas bort.

På de flesta operativsystem är cachekatalogen en tillfällig katalog. På iOS motsvarar cachekatalogen programbibliotekets katalog Caches. Filer i den här katalogen säkerhetskopieras inte till onlinelagringsutrymmet och kan tas bort av operativsystemet om enhetens tillgängliga lagringsutrymme är för litet. Du hittar mer information i ”Hantera säkerhetskopiering och cachning av filer” på sidan 679.

Peka på filsystemets rot

Adobe AIR 1.0 och senare

Metoden `File.getRootDirectories()` visar alla rotvolymer, till exempel C: och monterade volymer på en Windows-dator. I Mac OS och Linux returnerar den här metoden alltid datorns unika rotkatalog (katalogen "/"). Metoden `StorageVolumeInfo.getStorageVolumes()` ger mer detaljerad information om monterade lagringsvolymer (se ”Arbeta med lagringsvolymer” på sidan 690).

Obs! *Filsystemets rot kan inte läsas i Android. Ett File-objekt som refererar till katalogen med systemsökvägen ”/” returneras, men egenskaperna för det objektet har inte korrekta värden. `spaceAvailable` är till exempel alltid 0.*

Peka mot en viss katalog

Adobe AIR 1.0 och senare

Du kan peka File-objektet mot en viss katalog genom att ange egenskapen `nativePath` för File-objektet, som i följande exempel (i Windows):

```
var file:File = new File();
file.nativePath = "C:\\AIR Test";
```

Viktigt! Om du pekar på en explicit sökväg på det här sättet kan det innebära att koden inte fungerar på alla plattformar. Det föregående exemplet fungerar till exempel bara på Windows. Du kan använda de statiska egenskaperna i File-objektet, som `File.applicationStorageDirectory`, för att leta upp en katalog som fungerar på alla plattformar. Använd sedan metoden `resolvePath()` (se nästa avsnitt) för att navigera till en relativ sökväg.

Navigera till relativa sökvägar

Adobe AIR 1.0 och senare

Med metoden `resolvePath()` kan du hämta en sökväg som är relativ till en annan angiven sökväg. Följande kod ställer in ett File-objekt att peka mot underkatalogen ”AIR Test” till användarens hemkatalog:

```
var file:File = File.userDirectory;
file = file.resolvePath("AIR Test");
```

Du kan också använda egenskapen `url` för ett File-objekt om du vill peka det mot en katalog utifrån en URL-sträng, så här:

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

Mer information finns i ”Ändra File-sökvägar” på sidan 678.

Låta användarna bläddra för att välja en katalog

Adobe AIR 1.0 och senare

Klassen File omfattar metoden `browseForDirectory()`, som visar en systemdialogruta där användaren kan välja en katalog och tilldela den till objektet. Metoden `browseForDirectory()` är asynkron. File-objektet skickar en `select`-händelse om användaren väljer en katalog och klickar på knappen Öppna och skickar en `cancel`-händelse om användaren klickar på knappen Avbryt.

Följande kod gör så att användaren kan välja en katalog och returnerar katalogsökvägen när katalogen väljs:

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(e:Event):void {
    trace(file.nativePath);
}
```

Obs! *I Android stöds metoden `browseForDirectory()` inte. Om du anropar den här metoden har det ingen effekt alls, utan då skickas en cancel-händelse direkt. Om du vill att användaren ska kunna välja en katalog använder du en anpassad, programdefinierad dialogruta i stället.*

Peka mot den katalog som programmet anropas från

Adobe AIR 1.0 och senare

Du kan hämta den katalogplats som programmet anropas från genom att kontrollera egenskapen `currentDirectory` för InvokeEvent-objektet som skickas när programmet anropas. Mer information finns i ”Hämta kommandoradsargument” på sidan 882.

Peka ett File-objekt mot en fil

Adobe AIR 1.0 och senare

Du kan ställa in filen som ett File-objekt pekar mot på olika sätt.

Peka mot en explicit filsökväg

Adobe AIR 1.0 och senare

Viktigt! Om du pekar på en explicit sökväg kan det innebära att koden inte fungerar på alla plattformar. Sökvägen C:/foo.txt fungerar till exempel bara på Windows. Du kan använda de statiska egenskaperna i File-objektet, som `File.applicationStorageDirectory`, för att leta upp en katalog som fungerar på alla plattformar. Använd sedan metoden `resolvePath()` (se ”Ändra File-sökvägar” på sidan 678) för att navigera till en relativ sökväg.

Du kan använda egenskapen `url` för ett File-objekt om du vill peka det mot en fil eller katalog utifrån en URL-sträng, så här:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

Du kan också skicka URL:en till konstruktorfunktionen `File()`, så här:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

Egenskapen `url` returnerar alltid den URI-kodade versionen för URL:en (blanksteg ersätts till exempel med `"%20`):

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

Du kan också använda egenskapen `nativePath` för ett File-objekt om du vill ställa in en explicit sökväg. När följande kod körs på en Windows-dator ställs ett File-objekt in på att testa filen test.txt i AIR-underkatalogen Test på enhet C:

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

Du kan också skicka den här sökvägen till konstruktorfunktionen `File()`, så här:

```
var file:File = new File("C:/AIR Test/test.txt");
```

Använd snedstreck (/) som sökvägsavgränsare för egenskapen `nativePath`. På Windows kan du även använda omvänt snedstreck (\), men då kommer programmet inte att fungera på alla plattformar.

Mer information finns i ”Ändra File-sökvägar” på sidan 678.

Gå igenom filer i en katalog

Adobe AIR 1.0 och senare

Med metoden `getDirectoryListing()` för ett File-objekt kan du hämta en array med File-objekt som pekar mot filer och underkataloger på en katalogs rotnivå. Mer information finns i ”Gå igenom kataloger” på sidan 685.

Låta användarna bläddra för att välja en fil

Adobe AIR 1.0 och senare

Klassen File omfattar metoder som visar en systemdialogruta där användaren kan välja en fil och tilldela den till objektet:

- `browseForOpen()`
- `browseForSave()`
- `browseForOpenMultiple()`

Dessa metoder är asynkrona. Metoden `browseForOpen()` och `browseForSave()` skickar den valda händelsen när användaren väljer en fil (eller en målsökväg när det gäller browseForSave()). Med metoderna `browseForOpen()` och `browseForSave()` pekar mål-File-objektet mot de valda filerna. Metoden `browseForOpenMultiple`() skickar en `selectMultiple`-händelse när användaren väljer filer. Händelsen `selectMultiple` är av typen FileListEvent, som har en `files`-egenskap som är en array med File-objekt (som pekar mot de valda filerna).

Följande kod visar dialogrutan ”Öppna” för användaren där användaren kan välja en fil:

```
 var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

Om programmet har en annan webbläsardialogruta öppen när du anropar en bläddringsmetod, returneras ett Error-undantag.

Obs! *I Android kan bara bild-, video- och ljudfiler väljas med metoderna `browseForOpen()` och `browseForOpenMultiple()`. Dialogrutan browseForSave() visar dessutom bara mediefiler, även om användaren kan ange ett godtyckligt filnamn. Om icke-mediefiler ska kunna öppnas och sparas bör du använda anpassade dialogrutor i stället för de här metoderna.*

Ändra File-sökvägar

Adobe AIR 1.0 och senare

Du kan också ändra sökvägen för ett befintligt File-objekt genom att anropa metoden `resolvePath()` eller ändra egenskapen `nativePath` eller `url` för objektet, som i följande exempel (i Windows):

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("..");
trace(file2.nativePath); // C:\Documents and Settings\userName
var file3:File = File.documentsDirectory;
file3.nativePath += "/subdirectory";
trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4:File = new File();
file4.url = "file:///c:/AIR Test/test.txt";
trace(file4.nativePath); // C:\AIR Test\test.txt
```

När du använder egenskapen `nativePath` använder du snedstrecket (/) som katalogavgränsare. På Windows kan du även använda omvänt snedstreck (\), men du bör inte göra det eftersom det kan innebära att koden inte fungerar på alla plattformar.

URL-scheman som stöds i AIR

Adobe AIR 1.0 och senare

I AIR kan du använda följande URL-scheman när du definierar egenskapen `url` för ett File-objekt:

URL-schema	Beskrivning
file	Använd detta när du anger en sökväg som är relativ till filsystemets rot. Till exempel: `file:///c:/AIR Test/test.txt` URL-standarden anger att en fil-URL har formatet `file://<värd>/<sökväg>`. I vissa fall kan `<värd>` vara en tom sträng, som tolkas som ”datorn som URL:en tolkas från”. Därför innehåller fil-URL:er ofta tre snedstreck (///).
app	Använd detta när du anger en sökväg som är relativ till rotkatalogen för det installerade programmet (katalogen som innehåller filen application.xml för det installerade programmet). Följande sökväg pekar mot en bildunderkatalog till katalogen med det installerade programmet: `app:/images`
app-storage	Använd detta när du anger en sökväg som är relativ till programmets lagerkatalog. För varje installerat program definieras en unik programlagringskatalog i AIR, där du kan lagra data för just det programmet. Följande sökväg pekar till exempel på en prefs.xml-fil i en inställningsunderkatalog till programlagringskatalogen: `app-storage:/settings/prefs.xml`

Hantera säkerhetskopiering och cachning av filer

Adobe AIR 3.6 och senare, endast iOS och OS X

På vissa operativsystem, särskilt iOS och Mac OS X, har användaren möjlighet att automatiskt säkerhetskopiera programfiler till ett fjärrlagringsutrymme. På iOS finns det även begränsningar som rör huruvida filerna kan säkerhetskopieras och var filer med olika användningsområde kan sparas.

Nedan sammanfattas Apples riktlinjer för säkerhetskopiering och lagring av filer. Ytterligare information finns i efterföljande avsnitt.

- Om du vill ange att en fil inte behöver säkerhetskopieras, och (endast iOS) kan tas bort av operativsystemet om enhetens lagringsutrymme är för litet, sparar du filen i cachekatalogen (`File.cacheDirectory`). Detta är den rekommenderade lagringsplatsen på iOS och den bör användas för de flesta filer som kan återskapas eller hämtas igen.

- Om du vill ange att en fil inte behöver säkerhetskopieras, men inte heller ska tas bort av operativsystemet, sparar du filen i en av programkatalogerna, som programlagringskatalogen (`File.applicationStorageDirectory`) eller dokumentkatalogen (`File.documentsDirectory`). Ange File-objektets `preventBackup`-egenskap som `true`. Apple kräver detta för innehåll som kan återskapas eller hämtas igen, men som behövs för att ditt program ska fungera korrekt vid offlineanvändning.

Ange filer för säkerhetskopiering

För att spara på utrymmet som krävs för säkerhetskopior och minska belastningen på nätverket anger Apples riktlinjer för iOS- och Mac-program att endast filer som innehåller användarangivna data, eller data som inte kan återskapas eller hämtas igen, bör säkerhetskopieras.

Som standard säkerhetskopieras alla filer i programbiblioteksmapparna. På Mac OS X är detta programlagringskatalogen. På iOS inkluderar detta programlagringskatalogen, programkatalogen, skrivbordskatalogen, dokumentkatalogen och användarkatalogen (eftersom dessa kataloger är mappade till programbiblioteksmappar på iOS). Följaktligen säkerhetskopieras alla filer i dessa kataloger som standard till servern.

Om du sparar en fil som kan återskapas av programmet på någon av de här platserna bör du flagga filen så att operativsystemet inte säkerhetskopierar den. Du anger att en fil inte ska säkerhetskopieras genom att ange File-objektets `preventBackup`-egenskap som `true`.

Tänk på följande för iOS: En fil som finns i någon av programbiblioteksmapparna flaggas som en beständig fil som inte ska tas bort av operativsystemet, även om filens `preventBackup`-egenskap är `true`.

Hantera cachning och borttagning av filer

Apples riktlinjer för iOS-program anger att innehåll som kan återskapas i största möjliga mån bör kunna tas bort av operativsystemet, om enheten skulle få ont om lagringsutrymme.

På iOS flaggas filer i programbiblioteksmapparna (som programlagringskatalogen och dokumentkatalogen) som beständiga (permanenta) och tas inte bort av operativsystemet.

Spara filer, som kan återskapas av programmet och som kan tas bort utan problem vid utrymmesbrist, i programmets cachekatalog. Du kommer åt cachekatalogen med den statiska egenskapen `File.cacheDirectory`.

På iOS motsvarar cachekatalogen programmets cachekatalog (<Application Home>/Library/Caches). På andra operativsystem mappas den här katalogen till en likvärdig katalog. På Mac OS X mappas den till exempel även till katalogen Caches i programbiblioteket. På Android mappas cachekatalogen till programmets cachekatalog. På Windows mappas cachekatalogen till operativsystemets temp-katalog. På både Android och Windows är detta samma katalog som nås via ett anrop till File-klassens `createTempDirectory()`- och `createTempFile()`-metoder.

Hitta den relativa sökvägen mellan två filer

Adobe AIR 1.0 och senare

Med metoden `getRelativePath()` kan du söka efter den relativa sökvägen mellan två filer:

```
 var file1:File = File.documentsDirectory.resolvePath("AIR Test");
var file2:File = File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt
```

Den andra parametern för metoden `getRelativePath()`, parametern `useDotDot`, gör att syntaxen `..` kan returneras i resultat, vilket anger överordnade kataloger:

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3:File = File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..
trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

Skaffa kanoniska versioner av filnamn

Adobe AIR 1.0 och senare

Fil- och sökvägsnamn är inte skiftlägeskänsliga i Windows och Mac OS. Nedan pekar två File-objekt mot samma fil:

```
File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

Dokument- och katalognamnen har emellertid samma skiftläge. Nedan förutsätts det att det finns en mapp med namnet AIR Test i dokumentkatalogen:

```
var file:File = File.documentsDirectory.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
trace(file.nativePath); // ... AIR Test
```

Metoden `canonicalize()` konverterar `nativePath`-objektet så att rätt skiftläge används för fil- eller katalognamnet. Om användningen av gemener och versaler är det enda som skiljer olika filer åt i ett system som är skiftlägeskänsligt (t.ex. Linux) justerar `canonicalize()`-metoden sökvägen så att den matchar den första fil som hittas (i den ordning som fastställs av filsystemet).

Du kan också använda metoden `canonicalize()` om du vill konvertera korta filnamn (”8.3”-namn) till långa filnamn i Windows, som i följande exempel:

```
var path:File = new File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
trace(path.nativePath); // C:\AIR Test
```

Arbeta med paket och symboliska länkar

Adobe AIR 1.0 och senare

Olika operativsystem stöder paketfiler och filer med symboliska länkar:

Paket – I Mac OS kan kataloger definieras som paket och visas som en fil i Finder i Mac OS i stället för som en katalog.

Symboliska länkar – Symboliska länkar stöds i Mac OS, Linux och Windows Vista. Med hjälp av symboliska länkar kan en fil peka på en annan fil eller katalog på disken. Även om de påminner om varandra är symboliska länkar och alias inte samma sak. Ett alias rapporteras alltid som en fil (i stället för som en katalog), och om du läser eller skriver till aliaset eller genvägen påverkar detta inte den ursprungliga fil eller katalog som dessa pekar på. En symbolisk länk fungerar däremot precis som den fil eller katalog den pekar på. Den kan rapporteras som en fil eller katalog, och om du läser eller skriver till en symbolisk länk påverkar detta den fil eller katalog som länken pekar på, inte den symboliska länken. I Windows är dessutom egenskapen `isSymbolicLink` för ett File-objekt som refererar till en knutpunkt (används i NTFS-filsystemet) angiven som `true` .

Klassen File omfattar egenskaperna `isPackage` och `isSymbolicLink` som kontrollerar om ett File-objekt refererar till ett paket eller en symbolisk länk.

Följande kod itererar genom användarens skrivbordskatalog, och visar underkataloger som *inte* är paket:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !desktopNodes[i].isPackage)
    {
        trace(desktopNodes[i].name);
    }
}
```

Följande kod itererar genom användarens skrivbordskatalog, och visar filer och kataloger som *inte* är symboliska länkar:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        trace(desktopNodes[i].name);
    }
}
```

Metoden `canonicalize()` ändrar sökvägen för en symbolisk länk till att peka mot filen eller katalogen som länken refererar till. Följande kod itererar genom användarens skrivbordskatalog, och rapporterar sökvägarna som refereras av filer som är symboliska länkar:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode:File = desktopNodes[i] as File;
        linkNode.canonicalize();
        trace(linkNode.nativePath);
    }
}
```

Ta reda på ledigt utrymme på en volym

Adobe AIR 1.0 och senare

Egenskapen `spaceAvailable` för ett File-objekt anger det lediga utrymmet vid File-platsen, i byte. Följande kod kontrollerar det lediga utrymmet i programlagringskatalogen:

```
trace(File.applicationStorageDirectory.spaceAvailable);
```

Om filobjektet refererar till en katalog, visar egenskapen `spaceAvailable` utrymmet i den katalog som filen kan använda. Om filobjektet refererar till en fil, visar egenskapen `spaceAvailable` utrymmet som filen kan växa till. Om filplatsen inte finns, är egenskapen `spaceAvailable` inställd på 0. Om filobjektet refererar till en symbolisk länk, ställs egenskapen `spaceAvailable` in på det lediga utrymme som den symboliska länken pekar ut.

Vanligtvis är det tillgängliga utrymmet för en katalog eller fil samma som det tillgängliga utrymmet i volymen som innehåller katalogen eller filen. Kvoter och kataloggränser räknas dock med i tillgängligt utrymme.

Att lägga till en fil eller en katalog i en volym kräver generellt mer utrymme än den befintliga storleken på filen eller på innehållet i katalogen. Operativsystemet kan till exempel kräva mer utrymme för att lagra indexinformation. Eller också kan disksektorerna kräva mer utrymme. Tillgängligt utrymme ändras dynamiskt. Du kan alltså inte förvänta dig att tilldelas hela det rapporterade utrymmet för fillagring. Information om hur du skriver till filsystemet finns i ”Läsa och skriva filer” på sidan 691.

Metoden `StorageVolumeInfo.getStorageVolumes()` ger mer detaljerad information om monterade lagringsvolymer (se ”Arbeta med lagringsvolymer” på sidan 690).

Öppna filer med standardsystemprogrammet

Adobe AIR 2 och senare

I AIR 2 kan du öppna en fil med hjälp av det program som registrerats i operativsystemet för att öppna filen. Ett AIR-program kan till exempel öppna en DOC-fil med det program som registrerats för att öppna den. Använd metoden `openWithDefaultApplication()` för ett File-objekt för att öppna filen. Med följande kod öppnas till exempel en fil med namnet test.doc på användarens skrivbord med hjälp av standardprogrammet för DOC-filer:

```
var file:File = File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

Obs! *I Linux avgörs standardprogrammet för en fil av filens MIME-typ, inte genom filtillägget.*

Med följande kod kan användaren navigera till en mp3-fil och öppna den i standardprogrammet för att spela upp mp3-filer:

```
var file:File = File.documentsDirectory;
var mp3Filter:FileFilter = new FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);

function fileSelected(e:Event):void
{
    file.openWithDefaultApplication();
}
```

Du kan inte använda metoden `openWithDefaultApplication()` med filer som finns i programkatalogen.

Med AIR förhindras du att använda metoden `openWithDefaultApplication()` för att öppna vissa filer. I Windows förhindras du av AIR att öppna filer med vissa filtyper, till exempel EXE eller BAT. I Mac OS och Linux förhindrar AIR att du öppnar filer i vissa program. (Det gäller bland annat terminalen och AppletLauncher i Mac OS och csh, bash eller ruby i Linux.) Om du försöker öppna någon av dessa filer med metoden `openWithDefaultApplication()` resulterar det i ett undantag. En komplett lista över förhindrade filtyper finns under språkreferensposten för metoden `File.openWithDefaultApplication()`.

Obs! *Den här begränsningen finns inte i AIR-program som installerats med ett systemspecifikt installationsprogram (ett utökat skrivbordsprogram).*

Hämta information om filsystemet

Adobe AIR 1.0 och senare

Klassen File omfattar följande statiska egenskaper som tillhandahåller användbar information om filsystemet:

Egenskap	Beskrivning
`File.lineEnding`	Den teckensekvens för radslut som används av värdoperativsystemet. I Mac OS och Linux är detta radmatningstecknet. I Windows är detta vagnreturtecknet följt av radmatningstecknet.
`File.separator`	Värdoperativsystemets sökvägsavgränsare. I Mac OS och Linux är detta ett snedstreck (/). I Windows är det ett omvänt snedstreck (\).
`File.systemCharset`	Den standardkodning som används för filer av värdoperativsystemet. Detta hänför sig till den teckenuppsättning som används i operativsystemet, som motsvarar språket.

Klassen `Capabilities` omfattar också användbar systeminformation, som kan vara praktisk när du arbetar med filer:

Egenskap	Beskrivning
Capabilities.hasIME	Anger om spelaren körs i ett system som har (`true`) eller inte har (`false`) en IME (Input Method Editor) installerad.
Capabilities.language	Anger språkkoden för systemet som spelaren körs i.
Capabilities.os	Anger det nuvarande operativsystemet.

Obs! *Var försiktig när du använder `Capabilities.os` för att fastställa systemegenskaper. Om det finns en mer specifik egenskap för att fastställa en systemegenskap bör du använda den. Annars riskerar du att koden inte fungerar korrekt på alla plattformar. Titta på följande kod:*

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

Den här koden skapar problem i Linux. Det är bättre att bara använda egenskapen `File.separator`.

Arbeta med kataloger

Adobe AIR 1.0 och senare

I körningsversionen finns funktioner som gör att du kan arbeta med kataloger i det lokala filsystemet.

Mer information om hur du skapar File-objekt som pekar mot kataloger finns i ”Peka ett File-objekt mot en katalog” på sidan 673.

Skapa kataloger

Adobe AIR 1.0 och senare

Med metoden `File.createDirectory()` kan du skapa en katalog. Följande kod skapar en katalog med namnet AIR Test som är en underkatalog till användarens hemkatalog:

```
var dir:File = File.userDirectory.resolvePath("AIR Test");
dir.createDirectory();
```

Om katalogen finns utför metoden `createDirectory()` ingenting.

I vissa lägen skapar ett FileStream-objekt också kataloger när filer öppnas. Kataloger som saknas skapas när du instansierar en FileStream-instans med parametern `fileMode` för `FileStream()`-konstruktorn inställd på `FileMode.APPEND` eller `FileMode.WRITE`. Mer information finns i ”Arbetsflöde för att läsa och skriva filer” på sidan 691.

Skapa en tillfällig katalog

Adobe AIR 1.0 och senare

Klassen File omfattar en `createTempDirectory()`-metod, som skapar en katalog i den tillfälliga katalogmappen för systemet, som i följande exempel:

```
var temp:File = File.createTempDirectory();
```

Metoden `createTempDirectory()` skapar automatiskt en unik tillfällig katalog (så att du inte behöver fastställa en ny unik plats).

Du kan använda en tillfällig katalog om du tillfälligt behöver lagra filer som används i en programsession. Observera att det finns en `createTempFile()`-metod för att skapa nya, unika tillfälliga filer i den tillfälliga systemkatalogen.

Du kanske vill ta bort den tillfälliga katalogen innan programmet avslutas, eftersom den *inte* tas bort automatiskt på alla enheter.

Gå igenom kataloger

Adobe AIR 1.0 och senare

Med metoden `getDirectoryListing()` eller metoden `getDirectoryListingAsync()` för ett File-objekt kan du hämta en array med File-objekt som pekar mot filer och underkataloger i en katalog.

Följande kod visar innehållet i användarens dokumentkatalog (utan att undersöka underkatalogerna):

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
    trace(contents[i].name, contents[i].size);
}
```

När du använder den asynkrona versionen av metoden har `directoryListing`-händelseobjektet en `files`-egenskap som utgör arrayen med File-objekt som gäller för kataloger:

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

Kopiera och flytta kataloger

Adobe AIR 1.0 och senare

Du kan kopiera eller flytta en katalog, med samma metoder som när du kopierar eller flyttar en fil. Följande kod kopierar en katalog synkront:

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

När du anger true för parametern `overwrite` för metoden `copyTo()` tas alla filer och mappar i en befintlig målkatalog bort och ersätts med filerna och mapparna i källkatalogen (även om målfilen inte finns i källkatalogen).

Den katalog som du anger för parametern `newLocation` för metoden `copyTo()` anger sökvägen till den resulterande katalogen. Den anger *inte* den *överordnade* katalog som kommer att innehålla den resulterande katalogen.

Mer information finns i ”Kopiera och flytta filer” på sidan 688.

Ta bort innehåll i kataloger

Adobe AIR 1.0 och senare

Klassen File omfattar en `deleteDirectory()`-metod och en `deleteDirectoryAsync()`-metod. Dessa metoder tar bort kataloger; den första fungerar synkront medan den andra fungerar asynkront (se ”Grunderna för AIR-filer” på sidan 669). Båda metoderna har en `deleteDirectoryContents`-parameter (som använder ett booleskt värde). När den här parametern ställs in på `true` (standardvärdet är `false`) tar anropet till metoden bort alla kataloger som inte är tomma. Annars tas bara tomma kataloger bort.

Följande kod tar synkront bort underkatalogen AIR Test i användarens dokumentkatalog:

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

Följande kod tar asynkront bort underkatalogen AIR Test i användarens dokumentkatalog:

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

Metoderna `moveToTrash()` och `moveToTrashAsync()` ingår också, som du använder när du vill flytta en katalog till systemets papperskorg. Mer information finns i ”Flytta en fil till papperskorgen” på sidan 689.

Arbeta med -filer

Adobe AIR 1.0 och senare

Med API:et för AIR-filer kan du lägga till grundläggande funktioner för filinteraktion i programmet. Du kan till exempel läsa och skriva filer, kopiera och ta bort filer och så vidare. Eftersom programmet har tillgång till det lokala filsystemet bör du läsa igenom ”AIR-säkerhet” på sidan 1081 om du inte redan har gjort det.

Obs! *Du kan associera en filtyp med ett AIR-program (så att programmet öppnas när du dubbelklickar på en fil). Mer information finns i ”Hantera filassociationer” på sidan 890.*

Hämta filinformation

Adobe AIR 1.0 och senare

Klassen File omfattar följande egenskaper som tillhandahåller information om en fil eller katalog som ett File-objekt pekar mot:

File-egenskap	Beskrivning
creationDate	Det datum filen skapades på den lokala disken.
creator	Föråldrad – använd egenskapen `extension`. (Den här egenskapen rapporterar Macintosh-skapartypen för filen, som bara används i Mac OS-versioner tidigare än Mac OS X.)
downloaded	(AIR 2 och senare) Indikerar om referensfilen eller katalogen hämtades (från Internet) eller inte. Egenskapen har endast betydelse i operativsystem där hämtade filer kan flaggas: • Windows XP Service Pack 2 och senare och på Windows Vista • Mac OS 10.5 och senare
exists	Anger om den fil eller katalog som refereras finns.
extension	Ett filnamnstillägg är den del av namnet som följer efter den sista punkten (”.”). (Själva punkten är inte en del av tillägget.) Om det inte finns någon punkt i filnamnet är tillägget `null`.
icon	Ett Icon-objekt som innehåller de ikoner som har definierats för filen.
isDirectory	Anger om File-objektreferensen är en katalog.
modificationDate	Det datum då filen eller katalogen på den lokala disken senast ändrades.
name	Namnet på filen eller katalogen (inklusive filnamnstillägget, om det finns något) på den lokala disken.
nativePath	Den fullständiga sökvägen i värdoperativsystemets representation. Se ”Sökvägar till File-objekt” på sidan 671.
parent	Mappen som innehåller mappen eller filen som representeras av File-objektet. Den här egenskapen är `null` om File-objektet refererar till en fil eller katalog i filsystemets rot.
size	Filens storlek på den lokala disken, i byte.
type	Föråldrad – använd egenskapen `extension`. (I Macintosh anger den här egenskapen filtypen på fyra tecken, som bara används i Mac OS-versioner tidigare än Mac OS X.)
url	URL:en till filen eller katalogen. Se ”Sökvägar till File-objekt” på sidan 671.

Mer information om dessa egenskaper finns i avsnittet om File-klassen i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Kopiera och flytta filer

Adobe AIR 1.0 och senare

Klassen File omfattar två metoder för att kopiera filer eller kataloger: `copyTo()` och `copyToAsync()`. Klassen File omfattar två metoder för att flytta filer eller kataloger: `moveTo()` och `moveToAsync()`. Metoderna `copyTo()` och `moveTo()` fungerar synkront, och metoderna `copyToAsync()` och `moveToAsync()` fungerar asynkront (se ”Grunderna för AIR-filer” på sidan 669).

Om du vill kopiera eller flytta en fil ska du ställa in två File-objekt. Ett pekar mot filen som ska kopieras eller flyttas, och detta är objektet som anropar metoden copy eller move. Det andra pekar mot målsökvägen (resultatet).

Koden nedan kopierar en test.txt-fil från underkatalogen AIR Test i användarens dokumentkatalog till en fil med namnet copy.txt i samma katalog:

```
var original:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile:File = File.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

I det här exemplet ställs värdet på parametern `overwrite` för metoden `copyTo()` (den andra parametern) in på `true`. Genom att ställa in `overwrite` på `true` skrivs en befintlig målfil över. Den här parametern är valfri. Om du ställer in den på `false` (standardvärdet) skickar åtgärden en IOErrorEvent-händelse om målfilen finns (och filen kopieras inte).

”Async”-versionerna av metoderna copy och move fungerar asynkront. Använd metoden `addEventListener()` om du vill övervaka hur uppgiften genomförs eller felvillkoren, som i koden nedan:

```
var original = File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination:File = File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event:Event):void {
    trace(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event:IOErrorEvent):void {
    trace("I/O Error.");
}
```

Klassen File omfattar också metoderna `File.moveToTrash()` och `File.moveToTrashAsync()`, som flyttar en fil eller katalog till systemets papperskorg.

Ta bort en fil

Adobe AIR 1.0 och senare

Klassen File omfattar en `deleteFile()`-metod och en `deleteFileAsync()`-metod. Dessa metoder tar bort filer; den första fungerar synkront medan den andra fungerar asynkront (se ”Grunderna för AIR-filer” på sidan 669).

Följande kod tar synkront bort filen test.txt i användarens dokumentkatalog:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

Följande kod tar asynkront bort filen test.txt i användarens dokumentkatalog:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

Metoderna `moveToTrash()` och `moveToTrashAsync()` ingår också, som du använder när du vill flytta en fil eller katalog till systemets papperskorg. Mer information finns i ”Flytta en fil till papperskorgen” på sidan 689.

Flytta en fil till papperskorgen

Adobe AIR 1.0 och senare

Klassen File omfattar en `moveToTrash()`-metod och en `moveToTrashAsync()`-metod. Dessa metoder skickar en fil eller katalog till systemets papperskorg. Den första fungerar synkront medan den andra fungerar asynkront (se ”Grunderna för AIR-filer” på sidan 669).

Följande kod flyttar filen test.txt i användarens dokumentkatalog synkront till systemets papperskorg:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

Obs! *På operativsystem som saknar stöd för en återställningsbar papperskorg tas filerna bort direkt.*

Skapa en tillfällig fil

Adobe AIR 1.0 och senare

Klassen File omfattar en `createTempFile()`-metod, som skapar en fil i den tillfälliga katalogmappen för systemet, som i följande exempel:

```
var temp:File = File.createTempFile();
```

Metoden `createTempFile()` skapar automatiskt en unik tillfällig fil (så att du inte behöver fastställa en ny unik plats).

Du kan använda en tillfällig fil om du tillfälligt behöver lagra information som används i en programsession. Observera att det även finns en `createTempDirectory()`-metod för att skapa en unik tillfällig katalog i den tillfälliga systemkatalogen.

Du kanske vill ta bort den tillfälliga filen innan programmet avslutas, eftersom den *inte* tas bort automatiskt på alla enheter.

Arbeta med lagringsvolymer

Adobe AIR 2 och senare

I AIR 2 kan du identifiera när stora lagringsvolymer monteras eller avmonteras. Klassen StorageVolumeInfo definierar ett singletonobjekt av typen `storageVolumeInfo`. `StorageVolumeInfo.storageVolumeInfo`-objektet skickar en `storageVolumeMount`-händelse när lagringsvolymen monteras. Och den skickar en `storageVolumeUnmount`-händelse när en volym avmonteras. Dessa händelser definieras av klassen StorageVolumeChangeEvent.

***Obs!** På nyare Linux-distributioner skickar StorageVolumeInfo-objektet endast händelserna `storageVolumeMount` och `storageVolumeUnmount` för fysiska enheter och nätverksenheter som monterats på särskilda platser.*

Egenskapen `storageVolume` i klassen StorageVolumeChangeEvent är ett StorageVolume-objekt. Klassen StorageVolume definierar grundläggande egenskaper hos lagringsvolymen:

- `drive` – Volymetiketten för en Windows-enhet (`null` i andra operativsystem)
- `fileSystemType` – Typen av filsystem på lagringsvolymen (till exempel "FAT", "NTFS", "HFS" eller "UFS")
- `isRemoveable` – Om en volym är flyttbar (`true`) eller inte (`false`)
- `isWritable` – Om en volym är skrivbar (`true`) eller inte (`false`)
- `name` – Namnet på volymen
- `rootDirectory` – Ett File-objekt som motsvarar volymens rotkatalog

Klassen StorageVolumeChangeEvent innehåller även en `rootDirectory`-egenskap. Egenskapen `rootDirectory` är ett File-objekt som refererar till rotkatalogen för den lagringsvolym som har monterats eller avmonterats.

Egenskapen `storageVolume` för StorageVolumeChangeEvent-objektet är odefinierad (`null`) för en avmonterad volym. Men du kan komma åt egenskapen `rootDirectory` för händelsen.

Följande kod skickar namnet på och sökvägen till en lagringsvolym när den monteras:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_MOUNT, onVolumeMount);
function onVolumeMount(event:StorageVolumeChangeEvent):void
{
    trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

Följande kod skickar sökvägen till en lagringsvolym när den avmonteras:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_UNMOUNT, onVolumeUnmount);
function onVolumeUnmount(event:StorageVolumeChangeEvent):void
{
    trace(event.rootDirectory.nativePath);
}
```

Objektet `StorageVolumeInfo.storageVolumeInfo` omfattar en `getStorageVolumes()`-metod. Den här metoden returnerar en vektor över StorageVolume-objekt som motsvarar de lagringsvolymer som är monterade för tillfället. Följande kod visar hur namnen på och rotkatalogerna för alla monterade lagringsvolymer listas:

```
var volumes:Vector.<StorageVolume> = new Vector.<StorageVolume>;
volumes = StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (var i:int = 0; i < volumes.length; i++)
{
    trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

Obs! *På nyare Linux-distributioner returnerar metoden `getStorageVolumes()` objekt beroende på fysiska enheter och nätverksenheter som monterats på särskilda platser.*

Med metoden `File.getRootDirectories()` listas rotkatalogerna (se "Peka på filsystemets rot" på sidan 675. Men i StorageVolume-objekten (uppräknade av metoden `StorageVolumeInfo.getStorageVolumes()`) ges mer information om lagringsvolymerna.

Du kan använda egenskapen `spaceAvailable` i egenskapen `rootDirectory` i ett StorageVolume-objekt för att visa ledigt utrymme på en lagringsvolym. (Se "Ta reda på ledigt utrymme på en volym" på sidan 682

Fler hjälpavsnitt

StorageVolume

StorageVolumeInfo

Läsa och skriva filer

Adobe AIR 1.0 och senare

Med klassen FileStream kan AIR-program skriva och läsa i filsystemet.

Arbetsflöde för att läsa och skriva filer

Adobe AIR 1.0 och senare

Arbetsflödet för att läsa och skriva filer ser ut så här.

Initiera ett File-objekt som pekar på sökvägen.
File-objektet representerar sökvägen till filen som du vill arbeta med (eller till en fil som du skapar senare).

```
var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

I exemplet används egenskapen `File.documentsDirectory` och metoden `resolvePath()` för ett File-objekt för att initiera File-objektet. Det finns emellertid andra sätt att peka ett File-objekt mot en fil. Mer information finns i "Peka ett File-objekt mot en fil" på sidan 677.

Initiera ett FileStream-objekt.

Anropa metoden open() eller metoden openAsync() för FileStream-objektet.
Vilken metod du ska anropa beror på om du vill öppna filen för synkrona eller asynkrona åtgärder. Använd File-objektet som `file`-parameter för metoden open. För parametern `fileMode` anger du en konstant från klassen FileMode som anger hur du ska använda filen.

Följande kod initierar ett FileStream-objekt som används för att skapa en fil och skriva över befintliga data:

```
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);
```

Mer information finns i "Initialisera ett FileStream-objekt samt öppna och stänga filer" på sidan 693 och "Öppningslägen för FileStream" på sidan 692.

Om du öppnar filen asynkront (med metoden openAsync()) ska du lägga till och ställa in händelseavlyssnare för FileStream-objektet.
Dessa händelseavlyssnarmetoder svarar på händelser som skickas av FileStream-objektet i olika situationer. Det kan till exempel vara när data läses från filen, om ett I/O-fel påträffas eller när skrivningen av data har slutförts.

Mer information finns i ”Asynkron programmering och händelser som genereras av ett FileStream-objekt som öppnas asynkront” på sidan 697.

Inkludera kod för att läsa och skriva data, om det behövs.
Det finns många metoder för klassen FileStream som berör läsa och skriva. (Alla inleds med ”read” respektive ”write”.) Vilken metod som du ska välja för att läsa eller skriva data beror på formatet på data i målfilen.

Om data i målfilen till exempel är UTF-kodad text kan du använda metoderna `readUTFBytes()` och `writeUTFBytes()`. Om du vill hantera data som bytearrayer kan du använda metoderna `readByte()`, `readBytes()`, `writeByte()` och `writeBytes()`. Mer information finns i ”Dataformat och att välja read- eller write-metod” på sidan 697.

Om du öppnar filen asynkront måste du se till att det finns tillräckligt med data innan du anropar en läsmetod. Mer information finns i ”Läsbufferten och egenskapen bytesAvailable för ett FileStream-objekt” på sidan 695.

Om du vill kontrollera hur mycket ledigt diskutrymme som finns innan du skriver till en fil, kontrollerar du egenskapen spaceAvailable för File-objektet. Mer information finns i ”Ta reda på ledigt utrymme på en volym” på sidan 682.

Anropa metoden close() för FileStream-objektet när du har arbetat klart med filen.
Om du anropar metoden close() blir filen tillgänglig för andra program.

Mer information finns i ”Initialisera ett FileStream-objekt samt öppna och stänga filer” på sidan 693.

Ett exempelprogram som använder klassen FileStream för att läsa och skriva filer finns i följande artiklar på Adobe AIR Developer Center:

- Skapa en textfilsredigerare
- Skapa en textfilsredigerare
- Läsa och skriva från en XML-inställningsfil

Arbeta med FileStream-objekt

Adobe AIR 1.0 och senare

Klassen FileStream definierar metoder för att öppna, läsa och skriva filer.

Öppningslägen för FileStream

Adobe AIR 1.0 och senare

Metoderna `open()` och `openAsync()` för ett FileStream-objekt omfattar en `fileMode`-parameter, som definierar vissa egenskaper för en filström, bland annat följande:

- Möjligheten att läsa från filen
- Möjligheten att skriva till filen
- Om data alltid läggs till efter filslutet (vid skrivning)
- Vad som ska göras om filen inte finns (och när dess överordnade kataloger inte finns)

Nedan beskrivs olika fillägen (som du kan ange som parametern `fileMode` för metoden `open()` och `openAsync()`):

Filläge	Beskrivning
FileMode.READ	Anger att filen enbart är öppen för läsning.
FileMode.WRITE	Anger att filen enbart är öppen för skrivning. Om filen inte finns, skapas den när FileStream-objektet öppnas. Om filen finns, tas alla befintliga data bort.
FileMode.APPEND	Anger att filen enbart är öppen för tillägg. Filen skapas om den inte finns. Om filen finns, skrivs befintliga data inte över utan data börjar skrivas i slutet av filen.
FileMode.UPDATE	Anger att filen är öppen för läsning och skrivning. Om filen inte finns, skapas den. Ange det här läget för slumpmässig läsning/skrivning till filen. Du kan läsa från valfri position i filen. När du skriver till filen är det bara de byte som skrivs som skriver över befintliga byte (inga andra byte ändras).

Initialisera ett FileStream-objekt samt öppna och stänga filer

Adobe AIR 1.0 och senare

När du öppnar ett FileStream-objekt gör du det tillgängligt så att data kan läsas från och skrivas till en fil. Du öppnar ett FileStream-objekt genom att skicka ett File-objekt till metoden `open()` eller `openAsync()` för FileStream-objektet:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

Parametern `fileMode` (den andra parametern för metoderna `open()` och `openAsync()`) anger läget som filen ska öppnas i: för läsning, skrivning, tillägg eller uppdatering. Mer information finns i föregående avsnitt, ”Öppningslägen för FileStream” på sidan 692.

Om du använder metoden `openAsync()` för att öppna filen för asynkrona filåtgärder, ska du ställa in händelseavlyssnare till att hantera asynkrona händelser:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(IOErrorEvent.IO_Error, errorHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function completeHandler(event:Event):void {
    // ...
}

function progressHandler(event:ProgressEvent):void {
    // ...
}

function errorHandler(event:IOErrorEvent):void {
    // ...
}
```

Filen öppnas för synkrona eller asynkrona åtgärder, beroende på om du använder metoden `open()` eller `openAsync()`. Mer information finns i ”Grunderna för AIR-filer” på sidan 669.

Om du ställer in parametern `fileMode` på `FileMode.READ` eller `FileMode.UPDATE` i metoden open för FileStream-objektet, läses data till en läsbuffert så snart du öppnar FileStream-objektet. Mer information finns i ”Läsbufferten och egenskapen bytesAvailable för ett FileStream-objekt” på sidan 695.

Du kan anropa metoden `close()` för ett FileStream-objekt om du vill stänga den associerade filen, och göra den tillgänglig för andra program.

Egenskapen position för ett FileStream-objekt

Adobe AIR 1.0 och senare

Egenskapen `position` för ett FileStream-objekt fastställer var data läses eller skrivs vid nästa read- eller write-metod.

Före en read- eller write-åtgärd ska du ange egenskapen `position` till en giltig position i filen.

Följande kod skriver strängen `"hello"` (med UTF-kodning) vid position 8 i filen:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

När du först öppnar ett FileStream-objekt ställs egenskapen `position` in på 0.

Före en read-åtgärd måste värdet för `position` vara minst 0 och mindre än antalet byte i filen (som är de befintliga positionerna i filen).

Värdet på egenskapen `position` ändras bara under följande omständigheter:

- När du ställer in egenskapen `position` uttryckligen.
- När du anropar en read-metod.
- När du anropar en write-metod.

När du anropar en read- eller write-metod för ett FileStream-objekt ökas egenskapen `position` omedelbart med antalet byte som du läser eller skriver. Beroende på vilken read-metod som du använder ökas egenskapen `position` med antalet byte som du anger ska läsas eller antalet tillgängliga byte. När du anropar en read- eller write-metod efter en annan, börjar den läsa eller skriva vid den nya positionen.

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
trace(myFileStream.position); // 4200
```

Det finns emellertid ett undantag: för ett FileStream-objekt i tilläggsläge ändras inte egenskapen `position` efter ett anrop till en write-metod. (I tilläggsläge skrivs data alltid i slutet på filen, oberoende av värdet på egenskapen `position`.)

För en fil som öppnas för asynkrona åtgärder slutförs write-åtgärden inte förrän nästa kodrad körs. Du kan emellertid anropa flera asynkrona metoder i rad, så körs de i ordning:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(Event.CLOSE, closeHandler);
myFileStream.close();
trace("started.");

closeHandler(event:Event):void
{
    trace("finished.");
}
```

Trace-utdata för den här koden ser ut så här:

```
started.
finished.
```

Du *kan* ange `position`-värdet omedelbart efter att du har anropat en read- eller write-metod (eller när som helst), så att nästa read- eller write-åtgärd sker vid den positionen. Observera att följande kod ställer in egenskapen `position` direkt efter ett anrop till `writeBytes()`-åtgärden, och att `position` ställs in på det värdet (300) även efter att write-åtgärden är klar:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
trace(myFileStream.position); // 300
```

Läsbufferten och egenskapen bytesAvailable för ett FileStream-objekt

Adobe AIR 1.0 och senare

När ett FileStream-objekt med läsfunktioner (ett med parametern `fileMode` för metoden `open()` eller `openAsync()` inställd på `READ` eller `UPDATE`) öppnas lagrar körningsversionen data i en intern buffert. FileStream-objektet börjar att läsa data till bufferten så snart du öppnar filen (genom att anropa metoden `open()` eller `openAsync()` för FileStream-objektet).

För en fil som öppnas för synkrona åtgärder (med metoden `open()`) kan du alltid ställa in pekaren `position` till en giltig position (inom filens gränser) och börja läsa valfri datamängd (inom filens gränser), vilket visas i koden nedan (som förutsätter att filen innehåller minst 100 byte):

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

Oavsett om filen öppnas för synkrona eller asynkrona åtgärder läser read-metoderna alltid från ”tillgängliga” byte, som representeras av egenskapen `bytesAvalable`. När du läser synkront är alla byte i filen tillgängliga hela tiden. När du läser asynkront blir dessa byte tillgängliga vid den position som anges med egenskapen `position`, i en serie med asynkrona buffertfyllningar som signaleras med `progress`-händelser.

För filer som öppnas för *synkrona* åtgärder, ställs egenskapen `bytesAvailable` alltid in så att den representerar antalet byte från egenskapen `position` till slutet på filen (alla byte i filen är alltid tillgängliga för läsning).

För filer som öppnas för *asynkrona* åtgärder måste du se till att läsbufferten har förbrukat tillräckligt med data innan du anropar en read-metod. För en fil som öppnas synkront läggs data från filen till i bufferten allt eftersom read-åtgärden fortlöper, med början vid den `position` som anges när read-åtgärden startar, och egenskapen `bytesAvailable` ökar med varje byte som läses. Egenskapen `bytesAvailable` anger antalet tillgängliga byte med början vid den position som anges med egenskapen `position` till slutet på bufferten. FileStream-objektet skickar regelbundet en `progress`-händelse.

För en fil som öppnas asynkront skickar FileStream-objektet regelbundet `progress`-händelsen allt eftersom data blir tillgängliga i läsbufferten. Följande kod läser data till ett ByteArray-objekt, `bytes`, allt eftersom den läser till bufferten:

```
 var bytes:ByteArray = new ByteArray();
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function progressHandler(event:ProgressEvent):void
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

För en fil som öppnas asynkront kan bara data i läsbufferten läsas. Allt eftersom du läser data, tas de bort från läsbufferten. För read-åtgärder måste du se till att det finns data i läsbufferten innan du anropar read-åtgärden. Följande kod läser 8 000 byte med data från position 4 000 i filen:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
myFileStream.position = 4000;

var str:String = "";

function progressHandler(event:Event):void
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

Under en write-åtgärd läser inte FileStream-objektet några data till läsbufferten. När en write-åtgärd är klar (alla data i skrivbufferten har skrivits till filen) startar FileStream-objektet en ny läsbuffert (förutsatt att det tillhörande FileStream-objektet öppnades med läsfunktioner), och börjar läsa data till läsbufferten med början vid den position som anges med egenskapen `position`. Egenskapen `position` kan vara positionen för den sista byten som skrivs, eller vara en annan position om användaren anger ett annat värde för `position`-objektet efter write-åtgärden.

Asynkron programmering och händelser som genereras av ett FileStream-objekt som öppnas asynkront

Adobe AIR 1.0 och senare

När en fil öppnas asynkront (med metoden `openAsync()`) läses och skrivs filer asynkront. Allt eftersom data läses till läsbufferten och utdata skrivs, kan annan ActionScript-kod köras.

Det innebär att du måste registrera händelser som genereras av FileStream-objektet som har öppnats asynkront.

Genom att registrera `progress`-händelsen kan du bli meddelad när nya data blir tillgängliga för läsning, som i koden nedan:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progressHandler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Du kan läsa alla data genom att registrera `complete`-händelsen, som i koden nedan:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";
function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

På ungefär samma sätt som indata buffertlagras för att möjliggöra asynkron läsning, buffertlagras data som du skriver i en asynkron ström och de skrivs till filen asynkront. Allt eftersom data skrivs till en fil skickar FileStream-objektet regelbundet ett `OutputProgressEvent`-objekt. Ett `OutputProgressEvent`-objekt omfattar en `bytesPending`-egenskap som ställs in på antalet byte som finns kvar att skriva. Du kan registrera `outputProgress`-händelsen om du vill bli meddelad när bufferten verkligen skrivs till filen, kanske om du vill visa en förloppsdialogruta. I allmänhet behöver du emellertid inte göra detta. Du kan anropa metoden `close()` utan att behöva bekymra dig om byte som inte skrivs. FileStream-objektet fortsätter att skriva data och `close`-händelsen levereras efter att den sista byten har skrivits till filen och den underliggande filen stängs.

Dataformat och att välja read- eller write-metod

Adobe AIR 1.0 och senare

Alla filer består av en uppsättning byte på en disk. I ActionScript kan data från en fil alltid representeras som en ByteArray. Följande kod läser data från en fil till ett ByteArray-objekt med namnet `bytes`:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var bytes:ByteArray = new ByteArray();

function completeHandler(event:Event):void
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

Följande kod skriver på ett liknande sätt data från ett ByteArray-objekt med namnet `bytes` till en fil:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

Ofta vill du emellertid inte lagra data i ett ByteArray-objekt i ActionScript. Och ofta är filen i ett visst filformat.

Data i filen kan till exempel vara i ett textfilsformat, och du vill representera sådana data i ett String-objekt.

Därför omfattar klassen FileStream read- och write-metoder för läsning och skrivning av data till och från andra typer än ByteArray-objekt. Med till exempel metoden `readMultiByte()` kan du läsa data från en fil och lagra den i en sträng, som i följande kod:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Den andra parametern för metoden `readMultiByte()` anger textformatet som ActionScript använder för att tolka data (”iso-8859-1” i exemplet). Adobe AIR stöder kodning med vanliga teckenuppsättningar (se Teckenuppsättningar som stöds).

Klassen FileStream omfattar också metoden `readUTFBytes()`, som läser data från läsbufferten till en sträng med hjälp av teckenuppsättningen UTF-8. Eftersom tecknen i teckenuppsättningen UTF-8 har olika längd ska du inte använda `readUTFBytes()` i en metod som svarar på `progress`-händelsen, eftersom data i slutet av läsbufferten kan representera ett ofullständigt tecken. (Detta gäller också när du använder metoden `readMultiByte()` med teckenkodning med variabel längd.) Därför ska du läsa hela datauppsättningen när FileStream-objektet skickar händelsen `complete`.

Det finns också liknande write-metoder, `writeMultiByte()` och `writeUTFBytes()`, med vilka du kan arbeta med String-objekt och textfiler.

Metoderna `readUTF()` och `writeUTF()` (som inte ska blandas ihop med `readUTFBytes()` och `writeUTFBytes()`) läser och skriver också textdata till en fil, men de förutsätter att textdata föregås av information som anger längden på textdata, något som är ovanligt i vanliga textfiler.

Vissa UTF-kodade textfiler inleds med ett ”UTF-BOM”-tecken (byteordningsmarkör) som definierar graden av endian samt kodningsformatet (till exempel UTF-16 eller UTF-32).

Ett exempel på hur du läser och skriver till en textfil finns i ”Exempel: Läsa en XML-fil till ett XML-objekt” på sidan 700.

`readObject()` och `writeObject()` är praktiska när du vill lagra och hämta data för komplicerade ActionScript-objekt. Data kodas i AMF (ActionScript Message Format). Adobe AIR, Flash Player, Flash Media Server och Flex Data Services innehåller API:er för att arbeta med data i det här formatet.

Det finns några andra read- och write-metoder (till exempel `readDouble()` och `writeDouble()`). Om du använder dessa måste du emellertid se till att filformatet motsvarar formatet på de data som definieras av dessa metoder.

Filformat är ofta mer komplicerade än enkla textformat. En MP3-fil innehåller till exempel komprimerade data som bara kan tolkas med dekomprimerings- och avkodningsalgoritmer som bara gäller för MP3-filer. MP3-filer kan också inkludera ID3-koder som innehåller metatagginformation om filen (till exempel titel och artist för en låt). Det finns flera versioner av ID3-formatet, men den enklaste (ID3 version 1) beskrivs i avsnittet ”Exempel: Läsa och skriva data med direktåtkomst” på sidan 701.

Andra filformat (för bilder, databaser, programdokument och så vidare) har andra strukturer, och för att kunna arbeta med deras data i ActionScript måste du förstå hur dessa data struktureras.

Använda metoderna load() och save()

Flash Player 10 och senare, Adobe AIR 1.5 och senare

I Flash Player 10 introducerades metoderna `load()` och `save()` i klassen FileReference. Dessa metoder är också tillgängliga i AIR 1.5, och klassen File ärver metoderna från klassen FileReference. Syftet med dessa metoder är att användarna säkert ska kunna läsa in och spara filinformation i Flash Player. AIR-program kan emellertid också använda dessa metoder som ett enkelt sätt att läsa in och spara filer asynkront.

Med följande kod sparas exempelvis en sträng i en textfil:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
var str:String = "Hello.";
file.addEventListener(Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event:Event):void
{
    trace("Done.");
}
```

`save()`-metodens `data`-parameter kan tilldelas ett sträng-, XML- eller bytearrayvärde. Om argumentet är ett sträng- eller XML-värde sparas filen som en UTF-8-kodad textfil.

När det här kodexemplet körs visas en dialogruta där användaren väljer platsen för den sparade filen.

Följande kod läser in en sträng från en UTF-8-kodad textfil:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, loaded);
file.load();
var str:String;
function loaded(event:Event):void
{
    var bytes:ByteArray = file.data;
    str = bytes.readUTFBytes(bytes.length);
    trace(str);
}
```

Klassen FileStream tillhandahåller fler funktioner än metoderna `load()` och `save()`:

- Med FileStream-klassen kan du läsa och skriva data både synkront och asynkront.
- Med FileStream-klassen kan du skriva till en fil i flera steg.
- Med FileStream-klassen kan du öppna en fil för direktåtkomst (både läsning från och skrivning till valfritt avsnitt i filen).
- Med FileStream-klassen kan du ange vilken typ av filåtkomst du har till filen genom att definiera `open()`- eller `openAsync()`-metodens `fileMode`-parameter.
- Med FileStream-klassen kan du spara data till filer utan att visa en dialogruta av typen Öppna eller Spara för användaren.
- Du kan direkt använda andra typer än bytearrayer när du läser data med FileStream-klassen.

Exempel: Läsa en XML-fil till ett XML-objekt

Adobe AIR 1.0 och senare

I följande exempel visas hur du läser och skriver till en textfil som innehåller XML-data.

Om du vill läsa från filen initierar du File- och FileStream-objekten, anropar metoden `readUTFBytes()` för FileStream och konverterar strängen till ett XML-objekt:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.READ);
var prefsXML:XML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
fileStream.close();
```

Att skriva data till filen är lika enkelt som att ställa in rätt File- och FileStream-objekt och sedan anropa en write-metod för FileStream-objektet. Skicka strängversionen för XML-data till write-metoden så som i följande kod:

```
var prefsXML:XML = <prefs><autoSave>true</autoSave></prefs>;
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);

var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += prefsXML.toXMLString();

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

Dessa exempel använder metoderna `readUTFBytes()` och `writeUTFBytes()`, eftersom de förutsätter att filerna är i UTF-8-format. Om så inte är fallet, kanske du måste använda en annan metod (se ”Dataformat och att välja read- eller write-metod” på sidan 697).

I föregående exempel används FileStream-objekt som öppnats för synkrona åtgärder. Du kan också öppna filer för asynkrona åtgärder (som använder händelseavlyssnare för att svara på händelser). Följande kod visar hur du läser en XML-fil asynkront:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.addEventListener(Event.COMPLETE, processXMLData);
fileStream.openAsync(file, FileMode.READ);
var prefsXML:XML;

function processXMLData(event:Event):void
{
    prefsXML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
    fileStream.close();
}
```

Metoden `processXMLData()` anropas när hela filen läses till läsbufferten (när FileStream-objektet skickar händelsen `complete`). Den anropar metoden `readUTFBytes()` för att hämta en strängversion av läsdata, och skapar ett XML-objekt, `prefsXML`, baserat på den strängen.

Ett exempelprogram som visar dessa funktioner finns i Läsa och skriva från en XML-inställningsfil.

Exempel: Läsa och skriva data med direktåtkomst

Adobe AIR 1.0 och senare

MP3-filer kan inkludera ID3-koder, som är avsnitt i början eller slutet av filen som innehåller metadata som identifierar inspelningen. Själva ID3-kodformatet har olika revideringar. I det här exemplet beskrivs hur du läser och skriver från en MP3-fil som innehåller det enklaste ID3-formatet (ID3 version 1.0) med hjälp av ”direktåtkomst till fildata”, vilket innebär att det läser från och skriver till godtyckliga platser i filen.

En MP3-fil som innehåller en ID3 version 1-kod innehåller ID3-data i slutet av filen, i de sista 128 byten.

När du skaffar åtkomst till en fil för direkt läs-/skrivåtkomst är det viktigt att du anger `FileMode.UPDATE` som `fileMode`-parameter för metoden `open()` eller `openAsync()`:

```
var file:File = File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr:FileStream = new FileStream();
fileStr.open(file, FileMode.UPDATE);
```

Då kan du både läsa och skriva till filen.

När du öppnar filen kan du ställa in pekaren `position` på positionen 128 byte före slutet på filen:

```
fileStr.position = file.size - 128;
```

Den här koden ställer in egenskapen `position` på den här platsen i filen eftersom ID3 v1.0-formatet anger att ID3-koddata lagras i de sista 128 byten i filen. Specifikationen anger också följande:

- De första tre byten i koden innehåller strängen `"TAG"`.
- De nästa 30 tecknen innehåller titeln för MP3-spåret, som en sträng.
- De nästa 30 tecknen innehåller namnet på artisten, som en sträng.
- De nästa 30 tecknen innehåller namnet på albumet, som en sträng.
- De nästa fyra tecknen innehåller året, som en sträng.
- De nästa 30 tecknen innehåller kommentaren, som en sträng.
- Nästa byte innehåller en kod som anger spårets genre.
- Alla textdata är i formatet ISO 8859-1.

Metoden `id3TagRead()` kontrollerar data efter att de har lästs in (vid händelsen `complete`):

```
 function id3TagRead():void
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year:String = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode:String =  fileStr.readByte().toString(10);
    }
}
```

Du kan också skriva med direktåtkomst till filen. Du kan till exempel analysera variabeln `id3Title` för att se till att skiftläget är korrekt (med metoderna för klassen String), och sedan skriva en ändrad sträng med namnet `newTitle` till filen, så här:

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

Om du vill följa ID3 version 1-standarden ska längden på strängen `newTitle` vara 30 tecken, utfyllda vid slutet med teckenkoden 0 (`String.fromCharCode(0)`).

Kapitel 39: Lagra lokala data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd klassen SharedObject för att spara stora mängder data på klientdatorn. I Adobe AIR kan du även använda klassen EncryptedLocalStoreför att spara känslig information på den lokala datorn i ett AIR-program.

Du kan också läsa och skriva filer i filsystemet och (i Adobe AIR) komma åt lokala databasfiler. Du hittar mer information i ”Arbeta med filsystemet” på sidan 655 och ”Arbeta med lokala SQL-databaser i AIR” på sidan 716.

Det finns flera säkerhetsfaktorer som rör delade objekt. Du hittar mer information i ”Delade objekt” på sidan 1079 i ”Säkerhet” på sidan 1047.

Delade objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett delat objekt, som ibland kallas Flash-cookie, är en datafil som kan skapas på din dator av de webbplatser som du besöker. Delade objekt används ofta för att tillföra mervärde vid webbsurfning, så att du till exempel kan anpassa utseendet och funktionen hos en webbplats som du ofta besöker.

Om delade objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Delade objekt fungerar som cookies i en webbläsare. Använd klassen SharedObject för att lagra data på användarens lokala hårddisk och anropa dessa data under samma session eller i en senare session. Program har bara åtkomst till sina egna SharedObject-data och bara om de körs inom samma domän. Data skickas inte till servern och kan inte nås av andra program som körs på andra domäner, men de kan vara åtkomliga för program inom samma domän.

Delade objekt jämfört med cookies

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Cookies och delade objekt är mycket lika. Eftersom de flesta webbprogrammerare känner till hur cookies fungerar kan det vara praktiskt att jämföra cookies med lokala SharedObjects.

Cookies som uppfyller RFC 2109-standarden har i allmänhet följande egenskaper:

- De kan förfalla och gör det ofta som standard i slutet av en session.
- De kan inaktiveras av klienten på en platsspecifik grund.
- Det finns en gräns på totalt 300 cookies och maximalt 20 cookies per webbplats.
- De begränsas ofta till en storlek på 4 KB.
- De uppfattas ofta som ett säkerhetshot och inaktiveras därför ibland på klienten.
- De lagras på en plats som anges av klientens webbläsare.
- De överförs från klienten till servern med HTTP.

Delade objekt har å andra sidan följande egenskaper:

- De går inte ut.
- De begränsas som standard till en storlek på 100 KB.
- De kan lagra enkla datatyper (t.ex. sträng, matris och datum).
- De lagras på en plats som anges av programmet (i användarens hemkatalog).
- De överförs aldrig mellan klienten och servern.

Om klassen SharedObject

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Använd klassen SharedObject för att skapa eller ta bort delade objekt samt för att ta reda på den aktuella storleken på ett SharedObject-objekt som du använder.

Skapa ett delat objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill skapa ett SharedObject-objekt ska du använda metoden `SharedObject.getLocal()` som har följande syntax:

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

Exemplet nedan skapar ett delat objekt som kallas mySO:

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

En fil skapas på klientdatorn som kallas preferences.sol.

Termen *local* avser det delade objektets plats. I det här fallet lagrar Adobe® Flash® Player SharedObject-filen lokalt i klientens hemkatalog.

En ny katalog skapas för programmet och domänen i Flash Players sandlåda när du skapar ett delat objekt. En *.sol-fil skapas även som lagrar SharedObject-data. Filens standardplats är en underkatalog i användarens hemkatalog. Tabellen nedan visar standardplatserna för katalogen:

Operativsystem	Plats
Windows 95/98/ME/2000/XP	`c:/Documents and Settings/`*`username`*`/Application Data/Macromedia/Flash Player/#SharedObjects`
Windows Vista/Windows 7	`c:/Users/`*`username`*`/AppData/Roaming/Macromedia/Flash Player/#SharedObjects`
Macintosh OS X	`/Users/`*`username`*`/Library/Preferences/Macromedia/Flash Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol`
Linux/Unix	`/home/username/.macromedia/Flash_Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol`

#SharedObjects-katalogen nedan är en slumpmässigt namngiven katalog. Under den finns en katalog som motsvarar värdnamnet, programmets sökväg och slutligen *.sol-filen.

Om du t.ex. begär ett program kallat MyApp.swf på den lokala värddatorn i en underkatalog som kallas /sos, kommer *.sol-filen att lagras på följande plats i Windows XP:

```
c:/Documents and Settings/fred/Application Data/Macromedia/Flash
Player/#SharedObjects/KROKWXRK/#localhost/sos/MyApp.swf/data.sol
```

Obs! *Om du inte anger ett namn i `SharedObject.getLocal()`, kallas filen för undefined.sol.*

Som standard kan Flash spara lokalt upprepade SharedObject-objekt på upp till 100 KB per domän. Värdet kan konfigureras av användaren. När programmet försöker spara data till ett delat objekt som gör objektet större än 100 KB, visar Flash Player dialogrutan Lokal lagring där användaren kan tillåta eller neka mer lokal lagring för domänen som begär åtkomst.

Ange en sökväg

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda parametern *pathname* för att ange en plats för filen SharedObject. Filen måste vara i en underkatalog i domänens SharedObject-katalog. Om du t.ex. begär ett program på den lokala värden och anger följande:

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Skriver Flash Player SharedObject-filen i katalogen /#localhost (eller /localhost om programmet är offline). Det är praktiskt om du vill att mer än ett program på klienten ska ha åtkomst till det delade objektet. I detta fall kör klienten två Flex-program som båda anger en sökväg till det delade objektet i domänens rot och klienten har sedan tillgång till det delade objektet i båda programmen. Om du vill dela data mellan flera program utan upprepningar, kan du använda objektet LocalConnection.

Flash Player skapar inte en SharedObject-fil om du anger en katalog som inte finns.

Lägga till data i ett delat objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan lägga till en *.sol-fil för SharedObject med hjälp av egenskapen `data` för SharedObject-objektet. Använd följande syntax när du lägger till nya data i det delade objektet:

```
sharedObject_name.data.variable = value;
```

Exemplet nedan lägger till egenskaperna `userName`, `itemNumbers` och `adminPrivileges` samt deras värden i ett SharedObject:

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

När du har tilldelat värden till egenskapen `data`, måste du instruera Flash Player att skriva värdena till SharedObject-filen. Om du vill tvinga Flash Player att skriva värdena till SharedObject-filen kan du använda metoden `SharedObject.flush()`, på följande sätt:

```
mySO.flush();
```

Om du inte anropar `SharedObject.flush()` skriver Flash Player värdena till filen när programmet stängs. Det ger dock inte användaren möjlighet att öka utrymmet som Flash Player har tillgängligt för datalagring om datastorleken överstiger standardinställningarna. Därför är det bra rutin att anropa `SharedObject.flush()`.

När du använder metoden `flush()` för att skriva delade objekt till en användares hårddisk bör du vara noga med att kontrollera om användaren uttryckligen har inaktiverat lokal lagring med hjälp av inställningshanteraren i Flash Player (www.macromedia.com/support/documentation/se/flashplayer/help/settings_manager07.html), som framgår i följande exempel:

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

Lagra objekt i delade objekt

Du kan lagra enkla objekt som arrayer eller strängar i egenskapen `data` för ett SharedObject.

Exemplet nedan är en ActionScript-klass som definierar metoder som styr interaktionen med det delade objektet. Med dessa metoder kan användare lägga till och ta bort objekt från det delade objektet. Klassen lagrar en ArrayCollection som innehåller enkla objekt.

```
package {
    import mx.collections.ArrayCollection;
    import flash.net.SharedObject;

    public class LSOHandler {

        private var mySO:SharedObject;
        private var ac:ArrayCollection;
        private var lsoType:String;

        // The parameter is "feeds" or "sites".
        public function LSOHandler(s:String) {
            init(s);
        }

        private function init(s:String):void {
            ac = new ArrayCollection();
            lsoType = s;
            mySO = SharedObject.getLocal(lsoType);
            if (getObjects()) {
                ac = getObjects();
            }
        }

        public function getObjects():ArrayCollection {
            return mySO.data[lsoType];
        }

        public function addObject(o:Object):void {
            ac.addItem(o);
            updateSharedObjects();
        }

        private function updateSharedObjects():void {
            mySO.data[lsoType] = ac;
            mySO.flush();
        }
    }

}
```

Följande Flex-program skapar en instans av ActionScript-klassen för alla typer av delade objekt som det behöver. Det anropar sedan metoder för den klassen när användaren lägger till eller tar bort URL för bloggar eller webbplatser.

```
<?xml version="1.0"?>
<!-- lsos/BlogAggregator.mxml -->
<mx:Application
    xmlns:local="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initApp()"
    backgroundColor="#ffffff"
>
    <mx:Script>
        <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.utils.ObjectUtil;
        import flash.net.SharedObject;

        [Bindable]
        public var welcomeMessage:String;

        [Bindable]
        public var localFeeds:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var localSites:ArrayCollection = new ArrayCollection();

        public var lsofeeds:LSOHandler;
        public var lsosites:LSOHandler;

        private function initApp():void {
            lsofeeds = new LSOHandler("feeds");
            lsosites = new LSOHandler("sites");

            if (lsofeeds.getObjects()) {
                localFeeds = lsofeeds.getObjects();
            }
            if (lsosites.getObjects()) {
                localSites = lsosites.getObjects();
            }
        }

        // Adds a new feed to the feeds DataGrid.
        private function addFeed():void {
            // Construct an object you want to store in the
            // LSO. This object can contain any number of fields.
            var o:Object = {name:ti1.text, url:ti2.text, date:new Date()};
            lsofeeds.addObject(o);

            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method.
            localFeeds = lsofeeds.getObjects();

            // Clear the text fields.
            ti1.text = '';
            ti2.text = '';
        }
```

```
        // Removes feeds from the feeds DataGrid.
        private function removeFeed():void {
            // Use a method of ArrayCollection to remove a feed.
            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method. You do not need
            // to update it manually.
            if (myFeedsGrid.selectedIndex > -1) {

localFeeds.removeItemAt(myFeedsGrid.selectedIndex);
             }
        }

        private function addSite():void {
            var o:Object = {name:ti3.text, date:new Date()};
            lsosites.addObject(o);
            localSites = lsosites.getObjects();
            ti3.text = '';
        }

        private function removeSite():void {
            if (mySitesGrid.selectedIndex > -1) {

localSites.removeItemAt(mySitesGrid.selectedIndex);
            }
        }

        ]]>
    </mx:Script>

    <mx:Label text="Blog aggregator" fontSize="28"/>

    <mx:Panel title="Blogs">
        <mx:Form id="blogForm">
            <mx:HBox>
                <mx:FormItem label="Name:">
                    <mx:TextInput id="ti1" width="100"/>
                </mx:FormItem>
                <mx:FormItem label="Location:">
                    <mx:TextInput id="ti2" width="300"/>
                </mx:FormItem>
                <mx:Button id="b1" label="Add Feed" click="addFeed()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Feeds:">
                <mx:DataGrid
                    id="myFeedsGrid"
                    dataProvider="{localFeeds}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b2" label="Remove Feed" click="removeFeed()"/>
        </mx:Form>
    </mx:Panel>

    <mx:Panel title="Sites">
```

```
        <mx:Form id="siteForm">
            <mx:HBox>
                <mx:FormItem label="Site:">
                    <mx:TextInput id="ti3" width="400"/>
                </mx:FormItem>
                <mx:Button id="b3" label="Add Site" click="addSite()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Sites:">
                <mx:DataGrid
                    id="mySitesGrid"
                    dataProvider="{localSites}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b4" label="Remove Site" click="removeSite()"/>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

Lagra skrivna objekt i delade objekt

Du kan lagra skrivna ActionScript-instanser i delade objekt. Du kan göra det genom att anropa metoden `flash.net.registerClassAlias()` för att registrera klassen. Om du skapar en instans av klassen och lagrar den i datamedlemmen för det delade objektet och sedan läser ut objektet, får du en skriven instans. Som standard har SharedObject-egenskapen `objectEncoding` stöd för AMF3-kodning och packar upp den lagrade instansen från SharedObject-objektet. Den lagrade instansen behåller den typ som du specificerade när du anropade metoden `registerClassAlias()`.

(Endast iOS) Förhindra säkerhetskopior i molnet av lokalt delade objekt

Adobe AIR 3.7 och senare, endast iOS

Du kan ställa in egenskapen `SharedObject.preventBackup` för att kontrollera om lokala delade objekt kommer att säkerhetskopieras till iOS molnbaserade säkerhetskopieringstjänst. Apple kräver detta för innehåll som kan återskapas eller hämtas igen, men som behövs för att ditt program ska fungera korrekt vid offlineanvändning.

Skapa flera delade objekt

Du kan skapa flera delade objekt för samma Flex-program. Om du vill göra det måste du tilldela olika instansnamn till dem, så som exemplet nedan visar:

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

Det skapar en preferences.sol-fil och en history.sol-fil i Flex-programmets lokala katalog.

Skapa ett säkert SharedObject

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar ett lokalt eller fjärranslutet SharedObject med hjälp av `getLocal()` eller `getRemote()`, finns det en valfri parameter med namnet `secure` som avgör om åtkomsten till det här delade objektet är begränsat till SWF-filer som levereras via en HTTPS-anslutning. Om den här parametern är inställd på `true` och SWF-filen levereras via HTTPS, skapar Flash Player ett nytt säkert delat objekt eller hämtar en referens till ett befintligt säkert delat objekt. Det här säkra delade objektet kan bara läsas från eller skrivas till av SWF-filer som levereras över HTTPS och som anropar `SharedObject.getLocal()` med parametern secure inställd på `true` . Om den här parametern är inställd på `false` och SWF-filen levereras via HTTPS, skapar Flash Player ett nytt delat objekt eller hämtar en referens till ett befintligt delat objekt.

Det delade objektet kan läsas från eller skrivas till av SWF-filer som levereras via anslutningar utan HTTPS. Om SWF-filen levereras via en anslutning utan HTTPS och du försöker att ställa in parametern på `true`, går det inte att skapa ett nytt delat objekt (eller också misslyckas åtkomsten till ett befintligt säkert delat objekt), ett fel returneras och det delade objektet ställs in på `null`. Om du försöker att köra följande utdrag från en anslutning utan HTTPS, kommer metoden `SharedObject.getLocal()` att returnera ett fel:

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Oavsett parameterns värde räknas det delade objektet som skapats mot det totala diskutrymmet som tillåts för domänen.

Visa innehållet i ett delat objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Värden lagras i delade objekt i egenskapen `data`. Du kan göra en slinga över varje värde inom en delad objektinstans genom att använda en `for..in`-slinga, vilket visas i följande exempel:

```
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]);
}
```

Förstöra delade objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du vill förstöra ett SharedObject på klienten ska du använda metoden `SharedObject.clear()`. Det förstör inte kataloger i standardsökvägen för programmets delade objekt.

SharedObject-filen raderas från klienten i exemplet nedan:

```
public function destroySharedObject():void {
    mySO.clear();
}
```

SharedObject-exempel

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I följande exempel visas hur du kan spara enkla objekt, som exempelvis ett Date-objekt, i ett SharedObject-objekt utan att manuellt behöva serialisera och deserialisera dessa objekt.

Exemplet nedan börjar med att välkomna dig som en besökare för första gången. Programmet lagrar det aktuella datumet i ett delat objekt när du klickar på Log Out. Nästa gång du startar programmet eller uppdaterar sidan välkomnar programmet dig med en påminnelse om tidpunkten när du loggade ut.

Om du vill se hur programmet fungerar ska du starta programmet, klicka på Log Out och sedan uppdatera sidan. Programmet visar det datum och klockslag då du klickade på Log Out vid ditt föregående besök. Du kan när som helst ta bort den lagrade informationen genom att klicka på knappen Delete LSO.

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

Krypterad lokal lagringsplats

Med klassen EncryptedLocalStore (ELS) får du ett lokalt lagringsutrymme som du kan använda som ett litet cacheminne för ett programs egna data. ELS-data kan inte delas mellan program. Syftet med ELS är att program enkelt ska kunna lagra återskapade objekt, såsom inloggningsuppgifter och annan privat information. ELS-data ska inte anses som permanenta. Se även "Begränsningar för den krypterade lokala lagringsplatsen" och "God praxis" nedan.

***Obs!** Förutom den krypterade lokala lagringsplatsen tillhandahåller AIR kryptering för innehåll som lagras i SQL-databaser. Mer information finns i avsnittet "Använda kryptering med SQL-databaser" på sidan 760.*

Du bör använda den krypterade lokala lagringsplatsen för att cachelagra information som måste vara skyddad, till exempel inloggningsuppgifter för webbtjänster. Den krypterade lokala lagringsplatsen lämpar sig för att lagra information som måste hållas privat. Den skyddar emellertid inte data från processer som körs under samma användarkonto. Det lämpar sig således inte för att skydda hemliga programdata, som till exempel DRM- eller kryperingsnycklar.

På datorplattformar använder AIR DPAPI i Windows, KeyChain i Mac OS och KeyRing i Linux för att koppla den krypterade lokala lagringsplatsen till varje program och användare. På den krypterade lokala lagringsplatsen används AES-CBC 128-bitarskryptering.

På Android krypteras inte data lagrade med klassen EncryptedLocalStorage. I stället skyddas data av den användarsäkerhet som finns i operativsystemet. I operativsystemet Android tilldelas alla program ett separat användar-ID. Program har endast åtkomst till egna filer och filer skapade på publika platser (till exempel på utbytbara minneskort). Observera att på "öppna" Android-enheter HAR program som körs med rotprivilegier åtkomst till filer i andra program. På en "öppen" enhet kommer det krypterade lokala lagret inte att vara så skyddat som det är på en enhet som inte har "öppnats".

Informationen på den krypterade lokala lagringsplatsen är bara tillgänglig för AIR-programinnehåll i säkerhetssandlådan application.

Om du uppdaterar ett AIR-program kommer den uppdaterade versionen att få åtkomst till befintliga data i den krypterade lokala lagringsplatsen såvida inte:

- objekten lades till med parameternstronglyBound satt till `true`
- de befintliga och uppdaterade versionerna både är publicerade före AIR 1.5.3 och om uppdateringen är signerad med en migreringssignatur..

Begränsningar för den krypterade lokala lagringsplatsen

Data i den krypterade lokala lagringsplatsen skyddas av kontoinställningarna i användarens operativsystem. Andra kommer inte åt informationen på lagringsplatsen såvida de inte kan logga in som den aktuelle användaren. Informationen är emellertid inte skyddad från andra program som körs av en godkänd användare.

Eftersom användaren måste autentiseras för att de här attackerna ska kunna genomföras är användarens privata data fortfarande skyddade (om inte själva användarkontot har hackats). Data som programmet kan behöva hemlighålla för användaren, till exempel nyckelord för licensiering eller DRM, är dock inte skyddade. Den krypterade lokala lagringsplatsen är alltså inte rätt plats att spara den typen av information. Den lämpar sig bara för att lagra en användares privata data, till exempel lösenord.

Data kan av olika anledningar tas bort från den krypterade lokala lagringsplatsen. Användaren kan till exempel avinstallera programmet och ta bort den krypterade filen. Dessutom kan utgivar-ID ändras efter en uppdatering. Den krypterade lokala lagringsplatsen bör alltså behandlas som en privat cachelagringsplats och inte som en permanent datalagringsplats.

Parametern `stronglyBound` är föråldrad och ska inte sättas till `true`. Om du sätter parametern till `true` innebär det inte att du får något extra skydd för dina data. Samtidigt förlorar du åtkomsten till data när programmet uppdateras, även om du behåller utgivar-ID:t.

Den krypterade lokala lagringsplatsen kan prestera sämre om lagrade data överstiger 10 MB.

Om du avinstallerar ett AIR-program tas inte data som har lagrats på den krypterade lokala lagringsplatsen bort.

God praxis

De bästa sätten att använda den krypterade lokala lagringsplatsen är:

- Använd den krypterade lokala lagringsplatsen för att spara känsliga användardata, som lösenord (ange stronglyBound som false).
- Använd inte den krypterade lokala lagringsplatsen för att spara programhemligheter, som DRM-nycklar eller licensieringstokens.
- Erbjud ett sätt för programmet att återskapa data lagrade på lagringsplatsen om dessa data tas bort, till exempel genom att uppmana användaren att ange kontouppgifterna på nytt om det skulle behövas.

- Använd inte parametern `stronglyBound`.
- Migrera inte lagrade objekt under en uppdatering om du anger att `stronglyBound` ska vara `true`. Återskapa i stället data efter en uppdatering.
- Spara endast små mängder data. Använd en AIR SQL-databas med kryptering för större datamängder.

Fler hjälpavsnitt

flash.data.EncryptedLocalStore

Lägga till data på den krypterade lokala lagringsplatsen

Använd den statiska metoden `setItem()` i klassen EncryptedLocalStore för att spara data på den krypterade lokala lagringsplatsen. Data lagras i en hashtabell, med strängar som nycklar, där data lagras som bytearrayer.

Följande kod lagrar en sträng på den krypterade lokala lagringsplatsen:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes);
```

Den tredje parametern för metoden `setItem()`, parametern `stronglyBound`, är valfri. När den här parametern är inställd på `true` binds det lagrade objektet av den krypterade lokala lagringsplatsen till det lagrade AIR-programmets digitala signatur och bitar:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, false);
```

För ett objekt som lagras med `stronglyBound` inställt på `true` lyckas efterföljande anrop till `getItem()` bara om det anropande AIR-programmet är identiskt med det lagrande programmet (om inga datafiler i programkatalogen har ändrats). Om det anropande AIR-programmet inte är identiskt med det lagrande programmet, returneras ett Error-undantag när du anropar `getItem()` för ett starkt bundet objekt. Om du uppdaterar programmet går det inte att läsa starkt bundna data som tidigare har skrivits till den krypterade lokala lagringsplatsen. Att ställa in `true` för `stronglyBound` på mobila enheter ignoreras eftersom parametern alltid hanteras som `false`.

Om parametern `stronglyBound` anges som `false` (standard) behöver bara utgivar-ID:t förbli oförändrat för att programmet ska kunna läsa data. Bitarna i programmet kan ändras (och de måste signeras av samma utgivare), men de behöver inte vara exakt samma bitar som i det program som sparade informationen. Uppdaterade program med samma utgivar-ID som originalet har fortfarande åtkomst till informationen.

Obs! *I praktiken innebär det inget ökat dataskydd om du ställer in `stronglyBound` på `true`. En illvillig användare kan fortfarande göra ändringar i ett program för att få åtkomst till objekt som lagrats på den krypterade lokala lagringsplatsen. Data skyddas dessutom från externa, icke användarrelaterade hot lika effektivt oavsett om `stronglyBound` är inställt på `true` eller `false`. Därför bör du inte ställa in `stronglyBound` på `true`.*

Komma åt data på den krypterade lokala lagringsplatsen

Adobe AIR 1.0 och senare

Du kan hämta ett värde från den krypterade lokala lagringsplatsen genom att använda metoden `EncryptedLocalStore.getItem()`, som i följande exempel:

```
var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

Ta bort data från den krypterade lokala lagringsplatsen

Adobe AIR 1.0 och senare

Du kan ta bort ett värde från det krypterade lokala arkivet genom att använda metoden `EncryptedLocalStore.removeItem()`, som i följande exempel:

```
EncryptedLocalStore.removeItem("firstName");
```

Du kan radera alla data från den krypterade lokala lagringsplatsen genom att anropa metoden `EncryptedLocalStore.reset()`, som i följande exempel:

```
EncryptedLocalStore.reset();
```

Kapitel 40: Arbeta med lokala SQL-databaser i AIR

Adobe AIR 1.0 och senare

Med Adobe® AIR® kan du skapa och arbeta med lokala SQL-databaser. I körningen ingår en SQL-databasmotor med stöd för många standardfunktioner som använder databassystemet SQLite, baserat på öppen källkod. En lokal SQL-databas kan användas för lagring av lokala, beständiga data. Den kan t. ex. användas för programdata, användarinställningar, dokument eller andra typer av data som du vill att programmet sparar lokalt.

Om lokala SQL-databaser

Adobe AIR 1.0 och senare

Följande snabbstartartiklar hos Adobe Developer Connection innehåller korta förklaringar och kodexempel på hur du använder SQL-databaser:

- Arbeta asynkront med en lokal SQL-databas (Flex)
- Arbeta synkront med en lokal SQL-databas (Flex)
- Använda en krypterad databas (Flex)
- Arbeta asynkront med en lokal SQL-databas (Flash)
- Arbeta synkront med en lokal SQL-databas (Flash)
- Använda en krypterad databas (Flash)

I Adobe AIR ingår en SQL-baserad motor för relationsdatabaser som körs i körningen, med data som lagras lokalt i databasfiler på datorn som AIR-programmet körs på (till exempel på datorns hårddisk). Eftersom databasen körs och datafilerna lagras lokalt kan en databas användas med AIR-program, även utan nätverksanslutning. Tack vare körningens lokala SQL-databasmotor kan du enkelt lagra beständiga lokala programdata, särskilt om du redan har erfarenhet av SWL och relationsdatabaser.

Användningsområden för lokala SQL-databaser

Adobe AIR 1.0 och senare

Med funktionen för lokala SQL-databaser i AIR kan du lagra programdata på en användares lokala dator i flera olika syften. I Adobe AIR finns flera mekanismer för att lagra data lokalt. Varje mekanism har sina specifika fördelar. Nedan följer några exempel på hur du kan använda en lokal SQL-databas i ett AIR-program:

- När det gäller dataorienterade program (till exempel en adressbok) kan du använda en databas för att lagra data för huvudprogrammet.
- När det gäller dokumentorienterade program (i vilka användarna skapar dokument som de vill spara och eventuellt dela med andra) kan varje dokument sparas som en databasfil på valfri plats. (Observera dock att om databasen inte är krypterad kan databasfilen öppnas av ett AIR-program. Kryptering rekommenderas för dokument som innehåller potentiellt känslig information.)

- För nätverksmedvetna program kan du använda en databas för att lagra en lokal cache för programdata eller lagra data temporärt när en nätverksanslutning inte är tillgänglig. Du kan skapa en mekanism för att synkronisera den lokala databasen med ett datalager i nätverket.
- Oavsett vilken typ av program du har kan du använda en databas för att lagra enskilda användares programinställningar, såsom användaralternativ eller programinformation, till exempel fönsters storlek och placering.

Fler hjälpavsnitt

Christophe Coenraets: Employee Directory on AIR for Android (Medarbetarlista i AIR for Android)

Raymond Camden: jQuery and AIR - Moving from web page to application (jQuery och AIR – Flytta från webbsida till program)

Om AIR-databaser och -databasfiler

Adobe AIR 1.0 och senare

En lokal SQL-databas i Adobe AIR lagras som en enskild fil i datorns filsystem. Med SQL-databasmotorn som ingår i körningen kan du skapa och strukturera databasfiler samt ändra och hämta data från en databasfil. Körningen specificerar inte hur eller var databasdata lagras i filsystemet. Varje databas lagras i en enda fil. Du kan ange var databasfilen ska lagras i filsystemet. Du kan få åtkomst till en eller flera separata databaser (det vill säga separata databasfiler) från ett enskilt AIR-program. Eftersom varje databas lagras som en enskild fil i filsystemet kan du leta reda på din databas i enlighet med programmets utformning och operativsystemets begränsningar för filåtkomst. Varje användare kan ha en separat databasfil för sina specifika data, eller så kan en databasfil delas av alla programanvändare på en och samma dator. Eftersom data sparas lokalt på en enskild dator kan de inte delas automatiskt med användare på andra datorer. Motorn för den lokala SQL-databasen kan inte köra SQL-satser mot en fjärrdatabas eller serverbaserad databas.

Om relationsdatabaser

Adobe AIR 1.0 och senare

En relationsdatabas är en mekanism för lagring (och hämtning) av data på en dator. Data är ordnade i tabeller: rader representerar poster eller objekt, och kolumner (kallas ibland ”fält”) delar in varje post i enskilda värden. Ett adressboksprogram skulle till exempel kunna innehålla tabellen ”vänner”. Varje rad i tabellen skulle i så fall representera en enskild vän som lagras i databasen. Tabellens kolumner skulle representera data, till exempel förnamn, efternamn och födelsedatum. För varje vän-rad i tabellen lagras ett separat värde för varje kolumn.

Relationsdatabaser är utformade för att lagra komplexa data, där ett objekt är associerat med eller relaterat till objekt av en annan typ. I en relationsdatabas bör alla data som har en en-till-många-relation (där en enskild post kan vara relaterad till flera poster av en annan typ) delas upp i olika tabeller. Anta att du till exempel vill att ditt adressboksprogram ska lagra flera telefonnummer för varje vän. I så fall är detta en en-till-många-relation. Tabellen ”vänner” skulle innehålla all personlig information för varje vän. Den separata tabellen ”telefonnummer” skulle innehålla alla telefonnummer för alla vänner.

Förutom att lagra information om vänner och telefonnummer skulle data som anger relationen mellan de båda tabellerna behöva lagras i varje tabell, så att de enskilda vänposterna kan matchas med motsvarande telefonnummer. Dessa data kallas en primärnyckel, vilket är en unik identifierare som särskiljer varje rad i en tabell från andra rader i den tabellen. Primärnyckeln kan vara en ”naturlig nyckel”, vilket innebär att den är ett av dataobjekten som naturligt särskiljer varje post i tabellen. Om du vet att ingen av dina vänner i tabellen ”vänner” har samma födelsedatum, kan du använda kolumnen Födelsedatum som primärnyckel (en naturlig nyckel) för denna tabell. Om det inte finns någon naturlig nyckel kan du skapa en separat primär nyckel som t. ex. "vän-id" – ett artificiellt värde som programmet använder för att särskilja rader.

Med en primärnyckel kan du skapa relationer mellan flera tabeller. T. ex., om tabellen "vänner" har en kolumn som heter "vän-id" som innehåller ett unikt nummer för varje rad (varje vän). Den relaterade tabellen ”telefonnummer” kan struktureras med två kolumner: en kolumn med vän-id:t för vännen som telefonnumret tillhör och en kolumn med det faktiska telefonnumret. På så sätt kan alla telefonnummer för en enskild vän sparas i tabellen ”telefonnummer” (oavsett hur många telefonnummer vännen har) och länkas till den relaterade vännen med hjälp av primärnyckeln ”vän-id”. När en primärnyckel från en tabell används i en relaterad tabell för att ange kopplingen mellan posterna, kallas värdet i den relaterade tabellen för ”främmande nyckel”. Till skillnad från många databaser kan du inte skapa begränsningar för främmande nycklar i den lokala databasmotorn i AIR (begränsningar som automatiskt kontrollerar att ett infogat eller uppdaterat värde för en främmande nyckel har en motsvarande rad i primärnyckeltabellen). Relationer för främmande nycklar är en viktig del av strukturen för en relationsdatabas, och du använder främmande nycklar när du skapar relationer mellan tabeller i databasen.

Om SQL

Adobe AIR 1.0 och senare

SQL (Structured Query Language) används med relationsdatabaser för att ändra och hämta data. SQL är ett beskrivningsspråk snarare än ett procedurspråk. I stället för att ge datorn instruktioner för hur den ska hämta data anger du önskade data med en SQL-sats. Databasmotorn avgör hur dessa data hämtas.

Språket SQL har standardiserats av ANSI (American National Standards Institute). Den lokala SQL-databasen i Adobe AIR stöder huvuddelen av standarden SQL-92.

Mer specifika beskrivningar för SQL-språket i Adobe AIR finns i ”SQL-stöd i lokala databaser” på sidan 1109.

Om klasser i SQL-databaser

Adobe AIR 1.0 och senare

När du arbetar med lokala SQL-databaser i ActionScript 3.0 använder du instanser av dessa klasser i paketet flash.data:

Klass	Beskrivning
flash.data.SQLConnection	Gör att du kan skapa och öppna databaser (databasfiler) och innehåller även metoder för att utföra åtgärder på databasnivå och styra databastransaktioner.
flash.data.SQLStatement	Representerar en enskild SQL-sats (en fråga eller ett kommando) som körs på en databas, vilket innefattar definition av satstext och inställning av parametervärden.
flash.data.SQLResult	Med den här klassen kan du få information om eller körningsresultat för en sats, till exempel resultatrader från en `SELECT`-sats och antal rader som har påverkats av en `UPDATE`- eller `DELETE`-sats.

Använd följande klasser i paketet flash.data om du vill ha schemainformation som beskriver strukturen för en databas:

Klass	Beskrivning
flash.data.SQLSchemaResult	Fungerar som en behållare för databasschemaresultat som genereras genom anrop av metoden `SQLConnection.loadSchema()`.
flash.data.SQLTableSchema	Anger information som beskriver en enskild tabell i en databas.
flash.data.SQLViewSchema	Anger information som beskriver en enskild vy i en databas.
flash.data.SQLIndexSchema	Anger information som beskriver en enskild kolumn i en tabell eller vy i en databas.
flash.data.SQLTriggerSchema	Anger information som beskriver en enskild utlösare i en databas.

I paketet flash.data finns andra klasser som innehåller konstanter som används med klasserna SQLConnection och SQLColumnSchema:

Klass	Beskrivning
flash.data.SQLMode	Definierar en uppsättning konstanter som representerar möjliga värden för parametern `openMode` i metoderna `SQLConnection.open()` och `SQLConnection.openAsync()`.
flash.data.SQLColumnNameStyle	Definierar en uppsättning konstanter som representerar möjliga värden för egenskapen `SQLConnection.columnNameStyle`.
flash.data.SQLTransactionLockType	Definierar en uppsättning konstanter som representerar möjliga värden för alternativparametern i metoden `SQLConnection.begin()`.
flash.data.SQLCollationType	Definierar en uppsättning konstanter som representerar möjliga värden för egenskapen `SQLColumnSchema.defaultCollationType` och parametern `defaultCollationType` för `SQLColumnSchema()`-konstruktorn.

Följande klasser i paketet flash.events representerar händelser (och konstanter som stöds) som du använder:

Klass	Beskrivning
flash.events.SQLEvent	Definierar händelser som en SQLConnection- eller SQLStatement-instans skickar när någon av dess åtgärder har körts utan fel. Varje åtgärd har en tillhörande händelsetypskonstant definierad i klassen SQLEvent.
flash.events.SQLErrorEvent	Definierar händelser som en SQLConnection- eller SQLStatement-instans skickar när någon av dess åtgärder genererar ett fel.
flash.events.SQLUpdateEvent	Definierar händelsen som en SQLConnection-instans skickar när tabelldata i någon av dess anslutna databaser ändras när SQL-satsen `INSERT`, `UPDATE` eller `DELETE` körs.

Följande klasser i paketet flash.errors ger information om fel vid databasåtgärder:

Klass	Beskrivning
flash.errors.SQLError	Ger information om ett fel som uppkommit vid en databasåtgärd, inklusive vilken åtgärd som skulle utföras och orsaken till felet.
flash.errors.SQLErrorOperation	Definierar en uppsättning konstanter som representerar möjliga värden för klassen SQLErrors `operation`-egenskap, vilken anger databasåtgärden som resulterade i ett fel.

Om synkront och asynkront körningsläge

Adobe AIR 1.0 och senare

När du skriver kod för en lokal SQL-databas anger du att databasåtgärderna ska köras i antingen asynkront eller synkront körningsläge. I kodexemplen visas hur du utför varje åtgärd i respektive körningsläge, så att du kan använda det exempel som passar bäst för dig.

I asynkront körningsläge ger du körningen en instruktion. Sedan skickar körningen en händelse när din begärda åtgärd har slutförts eller misslyckats. Först anger du att databasmotorn ska utföra en åtgärd. Databasmotorn arbetar i bakgrunden medan programmet fortsätter att köras som vanligt. När åtgärden har slutförts (eller misslyckats) skickar databasmotorn en händelse. Din kod, som utlöses av händelsen, utför efterföljande åtgärder. Den här metoden har en klar fördel: databasåtgärderna utförs i bakgrunden medan huvudprogramkoden fortsätter att köras som vanligt. Programmet fortsätter att köras även om databasåtgärden tar lång tid att utföra. Viktigast av allt är att användaren kan fortsätta att interagera med programmet utan att skärmen fryser. Däremot kan kod för asynkrona åtgärder vara mer komplicerad att skriva än annan kod. Det kan till exempel vara komplicerat att skriva kod när flera beroende åtgärder måste delas upp i olika händelselyssnarmetoder.

Det är enklare att skriva koden för åtgärder som en enstaka stegsekvens (en uppsättning synkrona åtgärder) än en uppsättning åtgärder indelade i flera händelselyssnarmetoder. Med Adobe AIR kan du även köra databasåtgärder synkront. I synkront körningsläge körs åtgärderna inte i bakgrunden. I stället körs de i samma körningssekvens som all annan programkod. Du anger att databasmotorn ska utföra en åtgärd. Sedan pausas koden medan databasmotorn fortfarande utför sitt arbete. När åtgärden har slutförts fortsätter körningen med nästa rad i din kod.

Huruvida åtgärderna ska köras asynkront eller synkront anges på SQLConnection-nivå. Du kan inte köra vissa åtgärder eller satser synkront och andra asynkront med en enda databasanslutning. Du anger om en SQLConnection-instans ska arbeta i synkront eller asynkront körningsläge genom att anropa en SQLConnection-metod som öppnar databasen. Om du anropar `SQLConnection.open()` arbetar anslutningen i synkront körningsläge, och om du anropar `SQLConnection.openAsync()` arbetar anslutningen i asynkront körningsläge. När en SQLConnection-instans har anslutits till en databas via `open()` eller `openAsync()` är den kopplad till synkront eller asynkront körningsläge, om du inte stänger och sedan öppnar anslutningen till databasen på nytt.

Varje körningsläge har sina fördelar. De två lägena är till största delen lika, men det finns några skillnader som du bör tänka på när du arbetar i respektive läge. Mer information om detta och förslag på hur du kan arbeta i respektive läge finns i ”Använda synkrona och asynkrona databasåtgärder” på sidan 755.

Skapa och ändra en databas

Adobe AIR 1.0 och senare

Innan du kan lägga till eller hämta data med programmet måste en databas med tabeller ha definierats, och programmet måste kunna komma åt den. Nedan beskrivs vilka uppgifter du utför när du skapar en databas och datastrukturen i en databas. Du utför inte dessa uppgifter lika ofta som du infogar och hämtar data, men de är nödvändiga för de flesta program.

Fler hjälpavsnitt

Mind the Flex: Updating an existing AIR database (Uppdatera en befintlig AIR-databas)

Skapa en databas

Adobe AIR 1.0 och senare

Om du vill skapa en databasfil skapar du först en SQLConnection-instans. Du anropar dess `open()`-metod om du vill öppna den i synkront körningsläge, och dess `openAsync()` om du vill öppna den i asynkront körningsläge. Metoderna `open()` och `openAsync()` används för att öppna en anslutning till en databas. Om du skickar en File-instans som refererar till en filsökväg som inte finns för parametern `reference` (den första parametern), skapas en databasfil på den filsökvägen och en anslutning till databasen du nyss skapat öppnas med metoden `open()` eller `openAsync()`.

När du anropar metoden `open()` eller metoden `openAsync()` för att skapa en databas kan databasens filnamn vara vilket giltigt filnamn som helst, med vilket filnamnstillägg som helst. Om du anropar metoden `open()` eller `openAsync()` med värdet `null` för parametern `reference` skapas en ny IMDB (in-memory database) i stället för en databasfil på en disk.

I följande kod visas hur en databasfil skapas (en ny databas) i asynkront körningsläge. I det här fallet sparas databasfilen i "Peka mot programlagringskatalogen" på sidan 674 med filnamnet DBSample.db.

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database was created successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Obs! *Du kan använda File-klassen för att peka på en viss filsökväg, men om du gör det kan det leda till att program inte fungerar i olika plattformar. Sökvägen C:\Documents and Settings\joe\test.db fungerar exempelvis endast i Windows. Därför är det bäst att använda statiska egenskaper för File-klassen, till exempel* `File.applicationStorageDirectory`*, och metoden* `resolvePath()` *(som visas i det föregående exemplet). Mer information finns i* *"Sökvägar till File-objekt" på sidan 671.*

Om du vill köra åtgärder synkront anropar du metoden `open()` när du öppnar en databasanslutning med SQLConnection-instansen. I följande exempel visas hur du skapar och öppnar en SQLConnection-instans som kör åtgärder synkront:

```
import flash.data.SQLConnection;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile);
                    trace("the database was created successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Skapa databastabeller

Adobe AIR 1.0 och senare

När du skapar en tabell i en databas körs en SQL-sats på den databasen med samma process som du använder när du kör en SQL-sats, till exempel `SELECT`, `INSERT` och så vidare. När du skapar en tabell använder du en `CREATE TABLE`-sats, som innefattar definitioner av kolumner och begränsningar för den nya tabellen. Mer information om hur du kör SQL-satser finns i ”Arbeta med SQL-satser” på sidan 731.

I följande exempel visas hur du skapar en tabell med namnet ”employees” i en befintlig databasfil i asynkront körningsläge. Observera att koden i det här exemplet förutsätter att det finns en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till en databas.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(SQLEvent.RESULT, createResult);
createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event:SQLEvent):void
{
    trace("Table created");
}

function createError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                createStmt.addEventListener(SQLEvent.RESULT, createResult);
                createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

                createStmt.execute();
            }

            private function createResult(event:SQLEvent):void
            {
                trace("Table created");
            }

            private function createError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

I följande exempel visas hur du skapar en tabell med namnet "employees" i en befintlig databasfil i synkront körningsläge. Observera att koden i det här exemplet förutsätter att det finns en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till en databas.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                try
                {
                    createStmt.execute();
                    trace("Table created");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Ändra data i SQL-databaser

Adobe AIR 1.0 och senare

Vissa uppgifter utförs ofta när du arbetar med lokala SQL-databaser. Dessa åtgärder inkluderar anslutning till en databas, tillägg av data till tabeller och hämtning av data från tabeller i en databas. Det finns också flera saker du bör tänka på när du utför uppgifter, till exempel arbetar med datatyper och hanterar fel.

Observera att det också finns flera databasuppgifter som du inte utför så ofta, men som behöver göras innan du kan utföra de vanligare uppgifterna. Innan du kan ansluta till en databas och hämta data från en tabell måste du till exempel skapa databasen och skapa tabellstrukturen i databasen. Mindre vanliga initiala inställningsuppgifter behandlas i ”Skapa och ändra en databas” på sidan 720.

Du kan välja om du vill utföra databasåtgärder asynkront, vilket innebär att databasmotorn körs i bakgrunden och du får ett meddelande om att åtgärden har slutförts eller misslyckats via en händelse. Du kan också utföra dessa åtgärder synkront. I så fall utförs databasåtgärderna i följd och programmet (inklusive uppdateringar på skärmen) väntar tills åtgärderna har slutförts innan annan kod körs. Mer information om hur du arbetar i asynkront respektive synkront körningsläge finns i ”Använda synkrona och asynkrona databasåtgärder” på sidan 755.

Ansluta till en databas

Adobe AIR 1.0 och senare

Innan du kan utföra några databasåtgärder måste du först öppna en anslutning till databasfilen. En SQLConnection-instans används för att representera en anslutning till en eller flera databaser. Den första databasen som ansluts med en SQLConnection-instans kallas ”huvuddatabasen”. Den här databasen ansluts med metoden `open()` (för synkront körningsläge) eller `openAsync()` (för asynkront körningsläge).

Om du öppnar en databas med den asynkrona åtgärden `openAsync()`, aktiverar du SQLConnection-instansens `open`-händelse så att du vet när åtgärden `openAsync()` har slutförts. Aktivera SQLConnection-instansens `error`-händelse om du vill avgöra om åtgärden misslyckas.

I följande exempel visas hur du öppnar en befintlig databasfil för asynkron körning. Databasfilen heter DBSample.db och finns i användarens ”Peka mot programlagringskatalogen” på sidan 674.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, SQLMode.UPDATE);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile, SQLMode.UPDATE);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database opened successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

I följande exempel visas hur du öppnar en befintlig databasfil för synkron körning. Databasfilen heter DBSample.db och finns i användarens ”Peka mot programlagringskatalogen” på sidan 674.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, SQLMode.UPDATE);
    trace("the database opened successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile, SQLMode.UPDATE);
                    trace("the database opened successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Observera, att i anropet av metoden `openAsync()` i det asynkrona exemplet och i anropet av metoden `open()` i det synkrona exemplet, är det andra argumentet konstanten `SQLMode.UPDATE`. Om du anger `SQLMode.UPDATE` för den andra parametern (`openMode`) skickas ett fel i körningen om den angivna filen inte finns. Om du skickar `SQLMode.CREATE` för parametern `openMode` (eller om du utelämnar parametern `openMode`) görs ett försök att skapa en databasfil i körningen om den angivna filen inte finns. Om filen finns öppnas den emellertid, vilket är detsamma som om du använder `SQLMode.Update`. Du kan också ange `SQLMode.READ` för parametern `openMode` om du vill öppna en befintlig databas i skrivskyddat läge. I så fall kan data hämtas från databasen, men inga data kan läggas till, tas bort eller ändras.

Arbeta med SQL-satser

Adobe AIR 1.0 och senare

En enskild SQL-sats (en fråga eller ett kommando) representeras i körningen av ett SQLStatement-objekt. Så här skapar och kör du en SQL-sats:

Skapa en SQLStatement-instans.
SQLStatement-objektet representerar SQL-satsen i programmet.

```
var selectData:SQLStatement = new SQLStatement();
```

Ange vilken databas frågan ska köras mot.
Det gör du genom att ange SQLStatement-objektets `sqlConnection`-egenskap till SQLConnection-instansen som är ansluten till den önskade databasen.

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

Ange den faktiska SQL-satsen.
Skapa satstexten som en sträng och tilldela den till SQLStatement-instansens `text`-egenskap.

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

Definiera funktioner för att hantera resultatet av den åtgärd som har körts (endast asynkront körningsläge).
Använd metoden `addEventListener()` om du vill aktivera funktioner, till exempel avlyssnare för SQLStatement-instansens `result`- och `error`-händelser.

```
// using listener methods and addEventListener()

selectData.addEventListener(SQLEvent.RESULT, resultHandler);
selectData.addEventListener(SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event:SQLEvent):void
{
    // do something after the statement execution succeeds
}

function errorHandler(event:SQLErrorEvent):void
{
    // do something after the statement execution fails
}
```

Alternativt kan du ange lyssnarmetoder med hjälp av ett Responder-objekt. I så fall skapar du en Responder-instans och länkar lyssnarmetoderna till den.

```
// using a Responder (flash.net.Responder)

var selectResponder = new Responder(onResult, onError);

function onResult(result:SQLResult):void
{
    // do something after the statement execution succeeds
}

function onError(error:SQLError):void
{
    // do something after the statement execution fails
}
```

Om satstexten innehåller parameterdefinitioner tilldelar du värden för dessa parametrar.
Om du vill tilldela parametervärden använder du SQLStatement-instansens associativa array-egenskap `parameters`.

```
selectData.parameters[":param1"] = 25;
```

Kör SQL-satsen.
Anropa SQLStatement-instansens `execute()`-metod.

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

Om du använder en Responder i stället för händelselyssnare i asynkront körningsläge, skickar du Responder-instansen till `execute()`-metoden.

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

Specifika exempel som demonstrerar dessa steg finns i följande avsnitt:

”Hämta data från en databas” på sidan 735

”Infoga data” på sidan 745

”Ändra eller ta bort data” på sidan 751

Använda parametrar i programsatser

Adobe AIR 1.0 och senare

Med SQL-satsparametrar kan du skapa en återanvändbar SQL-sats. När du använder satsparametrar kan värdena inom satsen ändras (till exempel värden som läggs till i en `INSERT`-sats), men den grundläggande satstexten förblir oförändrad. Följaktligen ger det prestandafördelar att använda parametrar och det gör det enklare att koda ett program.

Om satsparametrar

Adobe AIR 1.0 och senare

I ett program används ofta en enskild SQL-sats flera gånger, med små variationer. Ta till exempel ett lagerhanteringsprogram där en användare kan lägga till nya lagerartiklar i en databas. En `INSERT`-sats (SQL) som lägger till data i databasen körs med den programkod som används för att lägga till lagerartikeln i databasen. Satsen körs dock med en liten variation varje gång. De faktiska värdena som infogas i tabellen skiljer sig åt eftersom de är specifika för den lagerartikel som läggs till.

Om du har en SQL-sats som används flera gånger med olika värden i satsen, är det bästa sättet att använda en SQL-sats att inkludera parametrar i stället för litteralvärden i SQL-texten. En parameter är en platshållare i satstexten som ersätts med ett faktiskt värde varje gång satsen körs. När du använder parametrar i en SQL-sats skapar du instansen SQLStatement som vanligt. Använd parameterplatshållare i stället för litteralvärden i den faktiska SQL-satsen som tilldelas till egenskapen `text`. Sedan definierar du värdet för varje parameter genom att ange ett värde för ett element i SQLStatement-instansens `parameters`-egenskap. Egenskapen `parameters` är en associativ array, vilket innebär att du anger ett visst värde med följande syntax:

```
statement.parameters[parameter_identifier] = value;
```

parameter_identifier är en sträng om du använder en namngiven parameter, eller ett heltalsindex om du använder en namnlös parameter.

Använda namngivna parametrar

Adobe AIR 1.0 och senare

En parameter kan vara namngiven eller namnlös. En namngiven parameter har ett specifikt namn som databasen använder för att matcha parametervärdet med platshållarens placering i satstexten. Ett parameternamn består av kolon (:) eller snabel-a (@) följt av ett namn:

```
:itemName
@firstName
```

I följande kod visas hur namngivna parametrar används:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

Använda namnlösa parametrar

Adobe AIR 1.0 och senare

Förutom att använda namngivna parametrar kan du även använda namnlösa parametrar. När du använder en namnlös parameter anger du en parameter i en SQL-sats med hjälp av ett frågetecken (?). Varje parameter tilldelas ett numeriskt index efter parametrarnas ordning i satsen. Den första parametern börjar på index 0. I följande exempel visas en version av det föregående exemplet, men med namnlösa parametrar:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

Fördelar med att använda parametrar

Adobe AIR 1.0 och senare

Att använda parametrar i en SQL-sats ger dig många fördelar:

Bättre prestanda Om du använder parametrar i en SQLStatement-instans blir körningen mer effektiv än om SQL-text skapas dynamiskt vid varje körning. Prestandaförbättringen beror på att satsen förbereds en gång och kan köras flera gånger med olika parametervärden, utan att SQL-satsen behöver kompileras om.

Explicit datatypning Parametrar används för att det senare ska gå att skriva in ersättningsvärden som var okända när SQL-satsen skapades. Det enda sättet att säkerställa lagringsklassen för ett värde som har skickats in i databasen är att använda parametrar. Om du inte använder parametrar görs ett försök i körningen att konvertera alla värden från textbeteckningen till en lagringsklass utifrån den associerade kolumnens typtillhörighet.

Mer information om lagringsklasser och kolumntillhörighet finns i ”Datatyper som stöds” på sidan 1130.

Ökad säkerhet Användningen av parametrar ger ett skydd mot så kallade SQL-injektionsangrepp. Vid ett SQL-injektionsangrepp tar sig en användare in i SQL-koden via en plats som är tillgänglig för användare (till exempel ett datainskrivningsfält). Om programkoden konstruerar en SQL-sats genom att direkt sammanfoga användarens indata med SQL-texten, körs användarens inmatade SQL-kod mot databasen. Nedan visas ett exempel på hur användares indata sammanfogas med SQL-texten. **Använd inte den här tekniken**:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;
```

SQL-injektionsangrepp förhindras om du använder satsparametrar i stället för att sammanfoga de värden användaren har angett i satstext. SQL-injektionen kan inte ske eftersom parametervärdena då uteslutande behandlas som ersättningsvärden i stället för att utgöra en del av den litterala satstexten. Följande alternativ rekommenderas i stället för tidigare beskrivning:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

Hämta data från en databas

Adobe AIR 1.0 och senare

Du hämtar data från en databas i två steg. Först kör du en `SELECT`-sats (SQL), som beskriver datamängden du vill hämta från databasen. Sedan kan du visa eller ändra hämtade data efter behov.

Köra en SELECT-sats

Adobe AIR 1.0 och senare

Om du vill hämta befintliga data från en databas använder du en SQLStatement-instans. Tilldela lämplig `SELECT`-sats (SQL) till instansens `text`-egenskap och anropa sedan dess `execute()`-metod.

Mer information om syntaxen för satsen `SELECT` finns i ”SQL-stöd i lokala databaser” på sidan 1109.

Följande exempel visar hur `SELECT`-satsen körs för att hämta data från en tabell som heter “products,” med asynkront körningsläge:

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}

function errorHandler(event:SQLErrorEvent):void
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
```

```
            selectStmt.execute();
        }

        private function resultHandler(event:SQLEvent):void
        {
            var result:SQLResult = selectStmt.getResult();

            var numResults:int = result.data.length;
            for (var i:int = 0; i < numResults; i++)
            {
                var row:Object = result.data[i];
                var output:String = "itemId: " + row.itemId;
                output += "; itemName: " + row.itemName;
                output += "; price: " + row.price;
                trace(output);
            }
        }

        private function errorHandler(event:SQLErrorEvent):void
        {
            // Information about the error is available in the
            // event.error property, which is an instance of
            // the SQLError class.
        }
    ]]>
  </mx:Script>
</mx:WindowedApplication>
```

Följande exempel visar hur `SELECT`-satsen körs för att hämta data från en tabell som heter “products,” med synkront körningsläge:

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}
catch (error:SQLError)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                try
                {
                    selectStmt.execute();

                    var result:SQLResult = selectStmt.getResult();
```

```
                var numResults:int = result.data.length;
                for (var i:int = 0; i < numResults; i++)
                {
                    var row:Object = result.data[i];
                    var output:String = "itemId: " + row.itemId;
                    output += "; itemName: " + row.itemName;
                    output += "; price: " + row.price;
                    trace(output);
                }
            }
            catch (error:SQLError)
            {
                // Information about the error is available in the
                // error variable, which is an instance of
                // the SQLError class.
            }
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

När satskörningen har slutförts i asynkront körningsläge skickar SQLStatement-instansen en `result`-händelse (`SQLEvent.RESULT`) som anger att satsen har körts utan fel. Om ett Responder-objekt i stället skickas som ett argument till metoden `execute()` anropas Responder-objektets resultathanterarfunktion. I synkront körningsläge pausas körningen tills `execute()`-åtgärden har slutförts. Sedan återupptas körningen på nästa rad i koden.

Få åtkomst till resultatdata för en SELECT-sats

Adobe AIR 1.0 och senare

När `SELECT`-satsen har körts är nästa steg att få åtkomst till de data som har hämtats. Du hämtar resultatdata från körning av en `SELECT`-sats genom att anropa SQLStatement-objektets `getResult()`-metod:

```
var result:SQLResult = selectStatement.getResult();
```

Metoden `getResult()` returnerar ett SQLResult-objekt. SQLResult-objektets egenskap `data` är en array som innehåller resultaten från satsen `SELECT`:

```
var numResults:int = result.data.length;
for (var i:int = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row:Object = result.data[i];
}
```

Varje datarad i resultatmängden `SELECT` blir en Object-instans i arrayen `data`. Detta objekt har egenskaper vilkas namn matchar resultatmängdens kolumnnamn. Egenskaperna innehåller värden från resultatmängdens kolumner. Anta till exempel att `SELECT`-satsen anger en resultatmängd med tre kolumner som heter ”itemId”, ”itemName” och ”price”. För varje rad i resultatmängden skapas en objektinstans med egenskaper som får namnen `itemId`, `itemName` och `price`. Dessa egenskaper innehåller värden från sina respektive kolumner.

I följande kodexempel definieras en SQLStatement-instans vars text är en `SELECT`-sats. Satsen hämtar rader som innehåller `firstName`- och `lastName`-kolumnvärden för alla rader i en tabell med namnet `employees`. I det här exemplet används asynkront körningsläge. När körningen har slutförts anropas metoden `selectResult()`, och du får åtkomst till resulterande datarader med `SQLStatement.getResult()` och visar dem med metoden `trace()`. Observera att koden i det här exemplet innehåller en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till databasen. I exemplet antas också att tabellen ”employees” redan har skapats och fyllts i med data.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event:SQLEvent):void
{
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                // register listeners for the result and error events
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

                // execute the statement
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void
            {
                // access the result data
                var result:SQLResult = selectStmt.getResult();

                var numRows:int = result.data.length;
                for (var i:int = 0; i < numRows; i++)
                {
                    var output:String = "";
                    for (var columnName:String in result.data[i])
                    {
                        output += columnName + ": " + result.data[i][columnName] + "; ";
                    }
                    trace("row[" + i.toString() + "]\t", output);
                }
            }

            private function selectError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

I följande kodexempel visas samma tekniker som i föregående exempel, men i synkront körningsläge. I exemplet definieras en SQLStatement-instans vars text är en `SELECT`-sats. Satsen hämtar rader som innehåller `firstName`- och `lastName`-kolumnvärden för alla rader i en tabell med namnet `employees`. Du får åtkomst till de resulterande dataraderna med `SQLStatement.getResult()` och visar dem med metoden `trace()`. Observera att koden i det här exemplet innehåller en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till databasen. I exemplet antas också att tabellen ”employees” redan har skapats och fyllts i med data.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();

    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                try
                {
                    // execute the statement
                    selectStmt.execute();

                    // access the result data
                    var result:SQLResult = selectStmt.getResult();

                    var numRows:int = result.data.length;
                    for (var i:int = 0; i < numRows; i++)
                    {
                        var output:String = "";
                        for (var columnName:String in result.data[i])
                        {
                            output += columnName + ": ";
                            output += result.data[i][columnName] + "; ";
                        }
                        trace("row[" + i.toString() + "]\t", output);
                    }
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Definiera datatypen för SELECT-resultatdata

Adobe AIR 1.0 och senare

Som standard skapas varje rad som returneras av en `SELECT`-sats som en objektinstans, där resultatmängdens kolumnnamn är objektets egenskapsnamn, och varje kolumns värde utgörs av värdet på den associerade egenskapen. Innan du kör en `SELECT`-sats (SQL) kan du dock ange SQLStatement-instansens `itemClass`-egenskap till en klass. När du anger egenskapen `itemClass` skapas varje rad som returneras av `SELECT`-satsen som en instans av den angivna klassen. I körningen tilldelas resultatkolumnvärden till egenskapsvärden genom att kolumnnamnen i `SELECT`-resultatmängden matchas med egenskaperna i klassen `itemClass`.

Varje klass som har tilldelats som ett `itemClass`-egenskapsvärde måste ha en konstruktor som inte kräver några parametrar. Klassen får dessutom bara ha en enda egenskap för varje kolumn som returneras av `SELECT`-satsen. Det ses som ett fel om en kolumn i `SELECT`-listan inte har något motsvarande egenskapsnamn i klassen `itemClass`.

Hämta SELECT-resultat i delar

Adobe AIR 1.0 och senare

Som standard hämtas alla rader i resultatmängden samtidigt vid en `SELECT`-satskörning. När satsen har slutförts bearbetar du vanligtvis de data som har hämtats på något sätt. Du kanske skapar objekt eller visar data på skärmen. Om satsen returnerar ett stort antal rader kan det vara krävande för datorn att bearbeta alla data samtidigt, vilket i sin tur kan medföra att användargränssnittet inte uppdateras (ritas om).

Du kan förbättra prestanda för programmet genom att ange att ett visst antal resultatrader ska returneras åt gången i körningen. På så sätt returneras initiala resultatdata snabbare. Det gör även att du kan dela in resultatraderna i mängder, så att användargränssnittet uppdateras efter att varje mängd rader har bearbetats. Observera att det endast är praktiskt att använda den här tekniken i asynkront körningsläge.

Om du vill hämta `SELECT`-resultat i delar anger du ett värde för `SQLStatement.execute()`-metodens första parameter (parametern `prefetch`). Parametern `prefetch` anger antalet rader som ska hämtas första gången satsen körs. När du anropar en SQLStatement-instans `execute()`-metod ska du ange ett `prefetch`-parametervärde så att endast det antalet rader hämtas:

```
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

Med satsen skickas en `result`-händelse som anger att den första mängden resultatrader är tillgänglig. Den resulterande SQLResult-instansens `data`-egenskap innehåller dataraderna och egenskapen `complete` anger om det finns ytterligare resultatrader att hämta. Om du vill hämta ytterligare resultatrader anropar du SQLStatement-instansens `next()`-metod. Precis som för metoden `execute()` används `next()`-metodens första parameter för att ange hur många rader som ska hämtas nästa gång result-händelsen skickas.

```
function selectResult(event:SQLEvent):void
{
    var result:SQLResult = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(SQLEvent.RESULT, selectResult);
        }
    }
}
```

SQLStatement skickar en `result`-händelse varje gång `next()`-metoden returnerar en efterföljande mängd resultatrader. Du kan alltså fortsätta att bearbeta resultaten med samma lyssnarfunktion (från `next()`-anrop) tills alla rader har hämtats.

Mer information finns i beskrivningarna för `SQLStatement.execute()`-metoden (beskrivningen av parametern `prefetch`) och `SQLStatement.next()`-metoden.

Infoga data

Adobe AIR 1.0 och senare

När du hämtar data till en databas kör du en `INSERT`-sats (SQL). När satskörningen har slutförts kan du få åtkomst till primärnyckeln för den nyligen infogade raden om nyckeln genererades av databasen.

Köra en INSERT-sats

Adobe AIR 1.0 och senare

Om du vill lägga till data i en tabell i en databas skapar och kör du en SQLStatement-instans vars text är en `INSERT`-sats (SQL).

I följande exempel används en SQLStatement-instans för att lägga till en datarad i den befintliga tabellen ”employees”. I det här exemplet visas hur du infogar data i asynkront körningsläge. Observera att koden i det här exemplet innehåller en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till en databas. I exemplet antas också att tabellen ”employees” redan har skapats.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event:SQLEvent):void
{
    trace("INSERT statement succeeded");
}

function insertError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
```

```
                "VALUES ('Bob', 'Smith', 8000)";
            insertStmt.text = sql;

            // register listeners for the result and failure (status) events
            insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
            insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

            // execute the statement
            insertStmt.execute();
        }

        private function insertResult(event:SQLEvent):void
        {
            trace("INSERT statement succeeded");
        }

        private function insertError(event:SQLErrorEvent):void
        {
            trace("Error message:", event.error.message);
            trace("Details:", event.error.details);
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

I följande exempel läggs en datarad till i den befintliga tabellen ”employees” i synkront körningsläge. Observera att koden i det här exemplet innehåller en SQLConnection-instans med namnet `conn` som redan har initierats och är ansluten till en databas. I exemplet antas också att tabellen ”employees” redan har skapats.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                try
                {
                    // execute the statement
                    insertStmt.execute();
                    trace("INSERT statement succeeded");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Hämta en databasgenererad primärnyckel för en infogad rad

Adobe AIR 1.0 och senare

När du har infogat en datarad i en tabell måste koden ofta innehålla en databasgenererad primärnyckel eller ett radidentifierarvärde för den nyligen infogade raden. När du till exempel har infogat en rad i en tabell kanske du vill lägga till rader i en relaterad tabell. I så fall måste du infoga primärnyckelvärdet som en främmande nyckel i den relaterade tabellen. Primärnyckeln för en nyligen infogad rad kan hämtas med hjälp av det SQLResult-objekt som är associerat med satskörningen. Det här är samma objekt som används för att få åtkomst till resultatdata efter att en `SELECT`-sats har körts. Liksom med SQL-satser skapas en SQLResult-instans när körningen av en `INSERT`-sats har slutförts. Du får åtkomst till SQLResult-instansen genom att anropa SQLStatement-objektets`getResult()`-metod om du använder en händelselyssnare eller använder synkront körningsläge. Alternativt kan du använda asynkront körningsläge och skicka en Responder-instans till `execute()`-anropet, så att SQLResult-instansen skickas som ett argument till resultathanterarfunktionen. Oavsett vilket alternativ du väljer har SQLResult-instansen egenskapen `lastInsertRowID`, som innehåller radidentifieraren för den senast infogade raden, om SQL-satsen som kördes är en `INSERT`-sats.

I följande exempel visas hur du får åtkomst till primärnyckeln för en infogad rad i asynkront körningsläge:

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
```

I följande exempel visas hur du får åtkomst till primärnyckeln för en infogad rad i synkront körningsläge:

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
catch (error:SQLError)
{
    // respond to the error
}
```

Observera att radidentifieraren kan vara (eller inte vara) värdet för kolumnen som har angetts som primärnyckelkolumn i tabelldefinitionen enligt följande regler:

- Om tabellen definieras med en primärnyckelkolumn vars tillhörighet (kolumndatatyp) är `INTEGER` innehåller egenskapen `lastInsertRowID` det värde som infogades i den raden (eller det värde som genererades i körningen om det är en `AUTOINCREMENT`-kolumn).
- Om tabellen definieras med flera primärnyckelkolumner (en sammansatt nyckel) eller med en enskild primärnyckelkolumn vars tillhörighet inte är `INTEGER`, genereras ett radidentifierarvärde för raden bakom scenen. Det genererade värdet är värdet för egenskapen `lastInsertRowID`.
- Värdet är alltid radidentifieraren för den senast infogade raden. Om en `INSERT`-sats aktiverar en utlösare, vilken i sin tur infogar en rad, kommer egenskapen `lastInsertRowID` att innehålla radidentifieraren för den senaste raden som infogats av utlösaren i stället för den rad som skapades av `INSERT`-satsen.

Om du som en konsekvens av de här reglerna vill ha en explicit definierad primärnyckelkolumn vars värde är tillgängligt efter ett `INSERT`-kommando via egenskapen `SQLResult.lastInsertRowID`, måste kolumnen definieras som en `INTEGER PRIMARY KEY`-kolumn. Men även om tabellen inte innehåller någon explicit `INTEGER PRIMARY KEY`-kolumn går det lika bra att använda den databasgenererade radidentifieraren som primärnyckel för tabellen när du definierar relationer med relaterade tabeller. Värdet för radidentifierarkolumnen är tillgängligt i alla SQL-satser när du använder något av kolumnnamnen `ROWID`, `_ROWID_` eller `OID`. Du kan skapa en kolumn för främmande nyckel i en relaterad tabell och använda radidentifierarvärdet som kolumnvärde för främmande nyckel på samma sätt som du

skulle ha gjort med en explicit deklarerad `INTEGER PRIMARY KEY`-kolumn. Om du använder en godtycklig primärnyckel i stället för en naturlig nyckel, och så länge du inte har något emot att primärnyckelvärdet genereras åt dig i körningen, blir det inte så stor skillnad om du använder en `INTEGER PRIMARY KEY`-kolumn eller den systemgenererade radidentifieraren som tabellens primärnyckel när du definierar en relation för främmande nyckel mellan två tabeller.

Mer information om primärnycklar och genererade radidentifierare finns i ”SQL-stöd i lokala databaser” på sidan 1109.

Ändra eller ta bort data

Adobe AIR 1.0 och senare

Processen för att köra andra databehandlingsåtgärder är identisk med processen som används för att köra en `SELECT`- eller `INSERT`-sats (SQL) som beskrivs i ”Arbeta med SQL-satser” på sidan 731. Du behöver bara byta ut SQL-satsen i SQLStatement-instansens `text`-egenskap:

- Använd en `UPDATE`-sats om du vill ändra befintliga data i en tabell.
- Använd en `DELETE`-sats om du vill ta bort en eller flera datarader från en tabell.

En beskrivning av dessa satser finns i ”SQL-stöd i lokala databaser” på sidan 1109.

Arbeta med flera databaser

Adobe AIR 1.0 och senare

Använd metoden `SQLConnection.attach()` om du vill öppna en anslutning till ytterligare en databas på en SQLConnection-instans som redan har en öppen databas. Du namnger den bifogade databasen med namnparametern i `attach()`-metodanropet. När du skriver satser som gör att databasen ändras, kan du använda det namnet i ett prefix (med formatet `database-name.table-name`) för att kvalificera alla tabellnamn i dina SQL-satser så att tabellen kan hittas i den namngivna databasen i körningen.

Du kan köra en enskild SQL-sats som innefattar tabeller från flera databaser som är anslutna till samma SQLConnection-instans. Om du skapar en transaktion på SQLConnection-instansen tillämpas den transaktionen för alla SQL-satser som körs med SQLConnection-instansen. Detta gäller oavsett vilken bifogad databas satsen körs på.

Du kan också skapa flera SQLConnection-instanser i ett program, vilka är anslutna till en eller flera databaser. Men om du använder flera anslutningar till samma databas måste du tänka på att en databastransaktion inte delas mellan olika SQLConnection-instanser. Om du ansluter till samma databasfil med flera SQLConnection-instanser kan du därför inte förlita dig på att båda anslutningarnas dataändringar tillämpas på förväntat sätt. Om till exempel två `UPDATE`- eller `DELETE`-satser körs mot samma databas via olika SQLConnection-instanser och ett programfel uppstår när en åtgärd körs, kan databasdata hamna i ett mellanläge som inte kan återställas och som kan påverka databasens integritet (och följaktligen även programmet).

Hantera databasfel

Adobe AIR 1.0 och senare

Hanteringen av databasfel fungerar i stort sett som annan hantering av körningsfel. Du bör skriva kod som är förberedd på olika fel som kan uppstå, och åtgärda felen i stället för att överlämna detta till körningen. De databasfel som kan uppstå kan i stort delas in i tre kategorier: anslutningsfel, SQL-syntaxfel och begränsningsfel.

Anslutningsfel

Adobe AIR 1.0 och senare

De flesta databasfel är anslutningsfel, och de kan uppstå vid alla typer av åtgärder. Även om det finns strategier för att förhindra anslutningsfel finns det sällan något enkelt sätt att göra en snygg återställning vid ett anslutningsfel om databasen är en kritisk del av programmet.

De flesta anslutningsfel har att göra med hur körningen interagerar med operativsystemet, filsystemet och databasfilen. Ett anslutningsfel uppstår till exempel om användaren inte har behörighet att skapa en databasfil på en viss plats i filsystemet. Du kan förhindra anslutningsfel med hjälp av följande strategier:

Använd användarspecifika databasfiler I stället för att använda en enda databasfil för alla användare som använder programmet på en dator, ger du varje användare en egen databasfil. Filen ska finnas i en katalog som är associerad med användarens konto. Den kan till exempel lagras i programmets lagringskatalog, i användarens dokumentmapp eller på användarens skrivbord.

Använd olika typer av användarkonton Testa att använda olika typer av användarkonton för programmet på olika operativsystem. Ta inte för givet att användaren har administratörsbehörighet på datorn. Ta inte heller för givet att personen som installerade programmet är den användare som kör programmet.

Använd olika filsökvägar Om du låter en användare ange var databasfilen ska kunna sparas eller välja vilken fil som ska öppnas, bör du överväga möjliga filsökvägar som användarna kan använda. Du bör dessutom överväga att definiera begränsningar för var användarna kan lagra (eller öppna) databasfiler. Du kan till exempel låta användarna endast öppna filer som finns på lagringsplatsen för användarkontot.

Om det uppstår ett anslutningsfel är det mest troligt att det sker vid det första försöket att skapa eller öppna databasen. Detta innebär att användaren inte kan göra några databasrelaterade åtgärder i programmet. För vissa typer av fel, till exempel fel som rör skrivskydd eller behörighet, är en möjlig återställningsteknik att kopiera databasfilen till en annan plats. Programmet kan kopiera databasfilen till en annan plats där användaren har behörighet att skapa eller skriva till filer, och använda den platsen i stället.

Syntaxfel

Adobe AIR 1.0 och senare

Ett syntaxfel uppstår när programmet försöker köra en SQL-sats som är felformaterad. Eftersom SQL-satser för lokala databaser skapas som strängar går det inte att utföra en SQL-syntaxkontroll vid kompileringen. Du måste kontrollera alla SQL-satsers syntax genom att köra dem. Använd följande strategier för att förhindra SQL-syntaxfel:

Testa alla SQL-satser noggrant När du utvecklar ditt program ska du, om möjligt, testa SQL-satserna separat innan du skriver dem som satstext i programkoden. Använd dessutom en kodtestningsmetod, till exempel enhetstest, för att skapa en uppsättning tester som utför alla möjliga alternativ och variationer i koden.

Använd satsparametrar och undvik sammanfogad (dynamiskt genererad) SQL Om du använder parametrar, och undviker dynamiskt skapade SQL-satser, används samma SQL-satstext varje gång en sats körs. Detta innebär att det blir mycket enklare att testa dina satser och begränsa antalet möjliga varianter. Om du måste generera en SQL-sats dynamiskt bör du hålla de dynamiska delarna av satsen på ett minimum. Verifiera också alla användarindata noggrant så att de inte orsakar syntaxfel.

Vid återställning av ett syntaxfel krävs komplex logik för att programmet ska kunna undersöka en SQL-sats och korrigera syntaxen. Om du följer de föregående riktlinjerna för hur du förhindrar syntaxfel kan potentiella källor till SQL-syntaxfel identifieras i körningen med hjälp av koden. Informera användaren om återställningsåtgärder vid syntaxfel. Ange vad som måste korrigeras för att satsen ska köras korrekt.

Begränsningsfel

Adobe AIR 1.0 och senare

Begränsningsfel uppstår när data läggs till i en kolumn med en `INSERT`- eller `UPDATE`-sats. Det här felet uppstår om nya data bryter mot någon av de definierade begränsningarna för tabellen eller kolumnen. Möjliga begränsningar:

Unique-begränsning Anger att det inte får finnas duplicerade värden i någon kolumn (gäller alla rader i en tabell). När flera kolumner kombineras i en unique-begränsning får inte heller kombinationen av värdena i dessa kolumner vara duplicerade. Alla rader måste med andra ord vara distinkta för en angiven unik kolumn (eller kolumner).

Primary key-begränsning När det gäller vilka data som en begränsning tillåter eller inte tillåter är primary key-begränsningen identisk med en unique-begränsning.

Not null-begränsning Anger att ett `NULL`-värde inte kan lagras i en enskild kolumn och att den kolumnen därför måste ha ett värde på varje rad.

Check-begränsning Gör att du kan ange en godtycklig begränsning för en eller flera tabeller. En vanlig Check-begränsning är en regel som definierar att en kolumns värde måste ligga inom vissa gränser (till exempel att en numerisk kolumns värde måste vara större än 0). En annan vanlig typ av Check-begränsning anger relationerna mellan kolumnvärden (till exempel att en kolumns värde måste skilja sig från värdet för en annan kolumn på samma rad).

Data type-begränsning (kolumntillhörighet) Körningen påtvingar datatypen för kolumnernas värden, och ett fel uppstår om det görs ett försök att lagra ett värde av felaktig typ i en kolumn. I många fall konverteras dock värden för att överensstämma med kolumnens deklarerade datatyp. Mer information finns i ”Arbeta med datatyper i databaser” på sidan 754.

I körningen tillämpas inte begränsningar på värden för främmande nycklar. Med andra ord krävs inte värden för främmande nycklar för att matcha ett befintligt primärnyckelvärde.

Utöver fördefinierade begränsningstyper stöder körningens SQL-motor även användning av utlösare. En utlösare liknar en händelsehanterare. Det är en fördefinierad uppsättning instruktioner som utförs vid en viss åtgärd. En utlösare kan till exempel definieras så att den aktiveras när data infogas i eller tas bort från en viss tabell. En möjlig användning för en utlösare är att undersöka dataändringar och utlösa ett fel om de angivna villkoren inte uppfylls. En utlösare kan alltså fylla samma syfte som en begränsning, och strategierna för förebyggande och återställande åtgärder vid begränsningsfel gäller även för utlösargenererade fel. Fel-id för utlösargenererade fel skiljer sig däremot från begränsningsfelens fel-id.

Vilken uppsättning begränsningar som ska tillämpas på en viss tabell avgör du medan du utformar ett program. Tänk på att utforma begränsningarna så att programmet kan vidta förebyggande och återställande åtgärder vid begränsningsfel. Begränsningsfel är dock mycket svåra att systematiskt förutspå och förebygga. Begränsningsfelen är svåra att förutspå eftersom de inte uppstår förrän programdata läggs till. Begränsningsfel uppstår med data som läggs till i en databas efter att den har skapats. Dessa fel är ofta ett resultat av relationen mellan nya data och data som redan finns i databasen. Med följande strategier kan du undvika många begränsningsfel:

Planera noggrant databasens struktur och begränsningar Syftet med begränsningarna är att tillämpa programregler och skydda integriteten för databasens data. När du planerar ditt program bör du fundera över hur du ska strukturera databasen så att den stöder programmet. I denna process ingår att identifiera regler för data, till exempel om vissa värden ska krävas, om ett värde ska ha ett standardvärde och om duplicerade värden ska vara tillåtna. Dessa regler ger en vägledning när du definierar begränsningar för databasen.

Ange kolumnnamn explicit Du kan skriva en `INSERT`-sats utan att du explicit anger vilka kolumner värden ska infogas i, men det medför onödiga risker. Genom att explicit namnge kolumnerna som värdena ska infogas i, blir det möjligt att använda automatiskt genererade värden, kolumner med standardvärden och kolumner som tillåter `NULL`-värden. På så sätt kan du dessutom se till att ett explicit värde infogas i alla `NOT NULL`-kolumner.

Använd standardvärden När du anger en `NOT NULL`-begränsning för en kolumn anger du ett standardvärde i kolumndefinitionen (om detta är möjligt). Det kan också finnas standardvärden i programkoden. Koden kan t.ex. kontrollera om en strängvariabel är `null` och tilldela den ett värde innan den används för att ange ett satsparametervärde.

Validera användarangivna data Kontrollera användarangivna data i förväg för att säkerställa att de uppfyller de krav som anges i begränsningarna (gäller särskilt begränsningarna `NOT NULL` och `CHECK`). Av naturliga skäl är begränsningen `UNIQUE` svårare att kontrollera eftersom en `SELECT`-fråga måste köras för att avgöra om aktuella data är unika.

Använd utlösare Du kan skriva en utlösare som validerar (och eventuellt ersätter) infogade data eller vidtar andra åtgärder för att korrigera ogiltiga data. Med hjälp av en sådan validering och korrigering kan du förhindra att det uppstår begränsningsfel.

Begränsningsfel är på många sätt svårare att förebygga än andra typer av fel. Som tur är finns flera strategier för att göra en återställning vid begränsningsfel utan att programmet blir instabilt eller oanvändbart:

Använd konfliktalgoritmer När du definierar en begränsning för en kolumn och när du skapar en `INSERT`- eller `UPDATE`-sats, kan du ange en konfliktalgoritm. En konfliktalgoritm definierar åtgärden som databasen vidtar när en överträdelse inträffar. Det finns flera möjliga åtgärder som databasmotorn kan vidta. Databasmotorn kan avsluta en enskild sats eller en hel transaktion. Den kan också ignorera felet. Den kan till och med ta bort gamla data och ersätta dem med data som koden försöker lagra.

Mer information finns i avsnittet "ON CONFLICT (konfliktalgoritmer)" i ”SQL-stöd i lokala databaser” på sidan 1109.

Ge korrigerande feedback En uppsättning begränsningar som kan påverka ett visst SQL-kommando kan identifieras i förväg. Du kan alltså förutse begränsningsfel som en sats kan orsaka. Med den kunskapen kan du skapa programlogik som åtgärdar ett begränsningsfel. Anta till exempel att ett program innehåller ett inmatningsformulär för att ange nya produkter. Om produktnamnskolumnen i databasen definieras med en `UNIQUE`-begränsning skulle i så fall ett begränsningsfel uppstå om en ny produktrad infogades i databasen. Programmet är därmed utformat för att förutse begränsningsfel. När felet inträffar får användaren en varning om att det angivna produktnamnet redan används och uppmanas att välja ett annat namn. Det går också att låta användaren visa information om den andra produkten med samma namn.

Arbeta med datatyper i databaser

Adobe AIR 1.0 och senare

När en tabell skapas i en databas definieras tillhörigheten, eller datatypen, för varje kolumn i tabellen av SQL-satsen som används för att skapa tabellen. Även om tillhörighetsdeklarationer kan utelämnas är det en god idé att explicit deklarera kolumntillhörigheten i `CREATE TABLE`-satser (SQL).

I regel returneras alla objekt som lagras i en databas med en `INSERT`-sats som en instans av samma datatyp när du kör en `SELECT`-sats. Datatypen för hämtade värden kan dock vara en annan beroende på tillhörigheten för databaskolumnen som värdet lagras i. Om datatypen för ett värde som lagras i en kolumn inte stämmer överens med kolumnens tillhörighet, försöker databasen konvertera värdet så att det matchar kolumnens tillhörighet. Om en databaskolumn till exempel deklareras med tillhörigheten `NUMERIC`, görs ett försök att konvertera infogade data i en numerisk lagringsklass (`INTEGER` eller `REAL`) innan dessa data lagras i databasen. Databasen genererar ett fel om dessa data inte kan konverteras. Regeln innebär att om strängen ”12345” infogas i en `NUMERIC`-kolumn, konverteras den automatiskt till heltalet 12345 innan den lagras i databasen. När den hämtas med en `SELECT`-sats, returneras värdet som en instans av en numerisk datatyp (till exempel Number) i stället för en stränginstans.

Det bästa sättet att undvika oönskad datatypskonvertering är att följa dessa två regler: Definiera först varje kolumn med en tillhörighet som stämmer överens med den typ av data som ska lagras i kolumnen. Infoga sedan endast värden vilkas datatyp stämmer överens med den definierade tillhörigheten. Det finns två fördelar med att följa dessa regler. När du infogar data konverteras de inte på ett oväntat sätt (vilket kanske hade medfört att de inte längre uppfyllde sitt avsedda syfte). När du hämtar data returneras de dessutom med sin ursprungliga datatyp.

Mer information om tillgängliga kolumntillhörighetstyper och hur du använder datatyper i SQL-satser finns i ”Datatyper som stöds” på sidan 1130.

Använda synkrona och asynkrona databasåtgärder

Adobe AIR 1.0 och senare

I tidigare avsnitt beskrevs vanliga databasåtgärder, till exempel hur du hämtar, infogar, uppdaterar och tar bort data samt hur du skapar en databasfil, tabeller och andra objekt i en databas. I exemplen visades hur du utför dessa åtgärder både asynkront och synkront.

I asynkront körningsläge instruerar du, som nämnts, databasmotorn att utföra en åtgärd. Databasmotorn arbetar sedan i bakgrunden medan programmet fortsätter att köras som vanligt. När åtgärden har slutförts skickar databasmotorn en händelse för att göra dig uppmärksam på detta. Den huvudsakliga fördelen med asynkron körning är att databasåtgärderna utförs i bakgrunden medan huvudprogramkoden fortsätter att köras som vanligt. Detta är särskilt värdefullt när en åtgärd tar lång tid att utföra.

I synkront körningsläge körs åtgärderna däremot inte i bakgrunden. Du anger att databasmotorn ska utföra en åtgärd. Sedan pausas koden medan databasmotorn fortfarande utför sitt arbete. När åtgärden har slutförts fortsätter körningen med nästa rad i din kod.

Du kan inte köra vissa åtgärder eller satser synkront och andra asynkront med en enda databasanslutning. Du anger om en SQLConnection-instans ska arbeta i synkront eller asynkront körningsläge när du öppnar anslutningen till databasen. Om du anropar `SQLConnection.open()` arbetar anslutningen i synkront körningsläge, och om du anropar `SQLConnection.openAsync()` arbetar anslutningen i asynkront körningsläge. När en SQLConnection-instans har anslutits till en databas via `open()` eller `openAsync()` är den kopplad till synkront eller asynkront körningsläge.

Använda synkrona databasåtgärder

Adobe AIR 1.0 och senare

Den faktiska koden du använder för att köra och svara på åtgärder i synkront körningsläge skiljer sig inte mycket från den kod du använder i asynkront körningsläge. Det finns två huvudskillnader mellan de två lägena. Den ena skillnaden är när du kör en åtgärd som är beroende av en annan åtgärd (till exempel `SELECT`-resultatrader eller primärnyckeln för en rad som har lagts till med en `INSERT`-sats). Den andra skillnaden gäller felhantering.

Skriva kod för synkrona åtgärder

Adobe AIR 1.0 och senare

Den huvudsakliga skillnaden mellan synkron och asynkron körning är att du i synkront läge skriver koden som en serie steg. I asynkron kod aktiverar du händelselyssnare och delar ofta in åtgärderna i olika lyssnarmetoder. När en databas är ansluten i synkront körningsläge kan du köra en serie databasåtgärder i följd inom ett enskilt kodblock. Detta visas i följande exempel:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

Som du kan se anropar du samma metoder för att utföra databasåtgärder oavsett om du använder synkron eller asynkron körning. De huvudsakliga skillnaderna mellan de två lägena är när du kör en åtgärd som är beroende av en annan åtgärd och när du hanterar fel.

Köra en åtgärd som är beroende av en annan åtgärd

Adobe AIR 1.0 och senare

När du använder synkront körningsläge behöver du inte skriva kod som lyssnar efter en händelse för att avgöra när en åtgärd har slutförts. I stället kan du anta att när en åtgärd i en kodrad har slutförts fortsätter nästa kodrad att köras. När du vill använda en åtgärd som är beroende av att en annan åtgärd först slutförs, behöver du alltså bara skriva koden direkt efter åtgärden den är beroende av. Om du till exempel vill att ett program ska påbörja en transaktion, köra en `INSERT`-sats, hämta primärnyckeln för den infogade raden, infoga primärnyckeln i en annan rad i en annan tabell och slutligen utföra transaktionen, skriver du koden som en serie satser. I följande exempel visas dessa åtgärder:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

Hantera fel med synkron körning

Adobe AIR 1.0 och senare

I synkront körningsläge behöver du inte lyssna efter en felhändelse för att avgöra om en åtgärd har misslyckats. I stället omsluter du kod som skulle kunna utlösa fel i en uppsättning `try..catch..finally`-kodblock. Du omsluter den felgenererande koden i `try`-blocket. Skriv åtgärderna som ska utföras som svar på varje typ av fel i separata `catch`-block. Placera all kod som du vill ska köras oavsett om det uppstår ett fel eller inte (det kan till exempel gälla stängning av en databasanslutning som inte längre behövs) i ett `finally`-block. I följande exempel visas hur du använder `try..catch..finally`-block för felhantering. I det här exemplet (som bygger på föregående exempel) läggs felhanteringskod till:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer:SQLStatement = new SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId:Number = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber:SQLStatement = new SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error:SQLError)
{
    // rollback the transaction
    conn.rollback();
}
```

Om asynkront körningsläge

Adobe AIR 1.0 och senare

Ett vanligt missförstånd om asynkron körning är att det inte går att påbörja en körning av en SQLStatement-instans om det redan körs en annan SQLStatement-instans mot samma databasanslutning. Detta antagande är felaktigt. Du kan inte ändra `text`-egenskapen för satsen medan en SQLStatement-instans körs. Men om du däremot använder en separat SQLStatement-instans för varje SQL-sats som du vill köra kan du anropa `execute()`-metoden för en SQLStatement-instans medan en annan SQLStatement-instans fortfarande körs, utan att det uppstår något fel.

När du kör databasåtgärder internt i asynkront körningsläge har varje databasanslutning (varje SQLConnection-instans) en egen kö eller lista med åtgärder som den har instruerats att utföra. Varje åtgärd körs i följd, i den ordning de har lagts till i kön. När du skapar en SQLStatement-instans och anropar dess `execute()`-metod läggs satskörningsåtgärden till i kön för anslutningen. Om ingen åtgärd redan körs på SQLConnection-instansen börjar satsen köras i bakgrunden. Anta att du i samma kodblock skapar en annan SQLStatement-instans och också anropar den metodens `execute()`-metod. Den andra satskörningsåtgärden läggs i så fall till efter den första satsen i kön. När den första satsen har slutförts körs nästa åtgärd i kön. Bearbetningen av efterföljande åtgärder i kön utförs i bakgrunden, även när `result`-händelsen för den första åtgärden skickas i huvudprogramkoden. Den här tekniken visas i följande kod:

```
// Using asynchronous execution mode
var stmt1:SQLStatement = new SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2:SQLStatement = new SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

Det finns en viktig sidoeffekt av att efterföljande satser i kön körs automatiskt. Om en sats är beroende av resultatet av en annan åtgärd kan du inte lägga till satsen i kön (du kan med andra ord inte anropa dess `execute()`-metod) förrän den första åtgärden har slutförts. Det beror på att när du väl har anropat den andra satsens `execute()`-metod kan du inte ändra satsens `text`- eller `parameters`-egenskaper. I så fall måste du vänta på händelsen som anger att den första åtgärden har slutförts innan du kan påbörja nästa åtgärd. Om du t. ex. vill köra en sats i överföringskontext är satskörningen beroende av åtgärden som öppnar transaktionen. När du har öppnat transaktionen genom att anropa metoden `SQLConnection.begin()` måste du vänta tills SQLConnection-instansen har skickat en `begin`-händelse. Du kan inte anropa SQLStatement-instansens `execute()`-metod förrän händelsen har skickats. I det här exemplet är det enklaste sättet att organisera programmet för att säkerställa att åtgärderna körs korrekt att skapa en metod som är registrerad som en lyssnare för `begin`-händelsen. Koden som ska anropa metoden `SQLStatement.execute()` placeras inom den lyssnarmetoden.

Använda kryptering med SQL-databaser

Adobe AIR 1.5 och senare

Samma lokala databasmotor används för alla Adobe AIR-program. Det betyder att alla AIR-program kan ansluta till, läsa från och skriva till en okrypterad databasfil. Från och med Adobe AIR 1.5 inkluderar AIR funktioner för att skapa och ansluta till krypterade databasfiler. När du använder en krypterad databas måste programmet som vill ansluta till databasen tillhandahålla korrekt krypteringsnyckel. Om fel krypteringsnyckel (eller ingen nyckel) anges kan programmet inte ansluta till databasen. Det innebär att programmet inte kan läsa data från databasen eller skriva till eller ändra data i databasen.

Om du vill använda en krypterad databas måste du skapa databasen som en krypterad databas. Om du har en befintlig krypterad databas kan du upprätta en anslutning till databasen. Du kan också ändra krypteringsnyckeln för en krypterad databas. Förutom när det gäller att skapa och ansluta till krypterade databaser, arbetar du på samma sätt med krypterade och okrypterade databaser. Exempelvis kör du SQL-uttryck på samma sätt oavsett om databasen är krypterad eller inte.

Användningsområden för en krypterad databas

Adobe AIR 1.5 och senare

Kryptering är användbart om du vill begränsa åtkomsten till informationen som lagras i en databas. Databaskrypteringsfunktionen i Adobe AIR kan användas för många ändamål. Här följer några exempel på situationer då det kan vara praktiskt att använda en krypterad databas:

- En skrivskyddad cache med privata programdata som hämtas från en server.
- Ett lokalt programarkiv för privata data som synkroniseras med en server (data skickas till och läses in från servern).
- Krypterade filer som används som filformatet för dokument som skapas och redigeras av programmet. Filerna kan vara privata för en användare, eller delas av alla som använder programmet.
- Annan användning av en lokal datalagringsplats, t.ex. de som beskrivs i avsnittet ”Användningsområden för lokala SQL-databaser” på sidan 716, där informationen måste skyddas från användare som har åtkomst till datorn eller databasfilerna.

Om du förstår när och varför det kan vara praktiskt att använda en krypterad databas blir det lättare att bestämma hur programmet ska struktureras. Detta kan exempelvis påverka hur krypteringsnyckeln för databasen skapas, hämtas eller lagras av programmet. Mer information finns i avsnittet ”Att tänka på när du använder kryptering med en databas” på sidan 764.

I stället för att använda en krypterad databas kan du skydda känsliga data med hjälp av den krypterade, lokala lagringsplatsen. Med den krypterade, lokala lagringsplatsen lagrar du ett enskilt ByteArray-värde med hjälp av en String-nyckel. Värdet är bara tillgängligt för det AIR-program som skapat värdet, och endast på den dator som det lagras på. Med den krypterade lokala lagringsplatsen behöver du inte skapa din egen krypteringsnyckel. Dessa anledningar gör att den krypterade lokala lagringsplatsen passar särskilt bra för att enkelt lagra ett enskilt värde eller en uppsättning värden som lätt kan kodas i en bytearray. En krypterad databas passar bäst för större dataupsättningar där strukturerad datalagring och datafrågor används. Mer information om hur du använder den krypterade lokala lagringsplatsen finns i avsnittet ”Krypterad lokal lagringsplats” på sidan 712.

Skapa en krypterad databas

Adobe AIR 1.5 och senare

Du kan bara använda en krypterad databas om databasen krypterades när den skapades. En okrypterad databas kan inte krypteras senare. På motsvarande sätt kan en krypterad databas inte dekrypteras senare. Om det behövs kan du ändra krypteringsnyckeln för en krypterad databas. Mer information finns i avsnittet ”Ändra krypteringsnyckeln för en databas” på sidan 763. Om du har en befintlig databas som inte är krypterad och du vill använda databaskryptering kan du skapa en ny krypterad databas och kopiera den befintliga tabellstrukturen och data till den nya databasen.

Du skapar en krypterad databas på praktiskt taget samma sätt som en okrypterad databas. Mer information finns i avsnittet ”Skapa en databas” på sidan 721. Först skapar du en SQLConnection-instans som representerar anslutningen till databasen. Du skapar databasen genom att anropa SQLConnection-objektets `open()`-metod eller `openAsync()`-metod och anger en fil som inte finns än för databasplatsen. Den enda skillnaden när du skapar en krypterad databas är att du anger ett värde för `encryptionKey`-parametern (`open()`-metodens femte parameter och `openAsync()`-metodens sjätte parameter).

Ett giltigt `encryptionKey`-parametervärde är ett ByteArray-objekt som innehåller exakt 16 byte.

I följande exempel visas hur en krypterad databas skapas. För enkelhetens skull är krypteringsnyckeln hårdkodad i programkoden i de här exemplen. Den här tekniken rekommenderas emellertid inte eftersom den inte är säker.

```
var conn:SQLConnection = new SQLConnection();

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, SQLMode.CREATE, false, 1024, encryptionKey);
```

Ett exempel som illustrerar en rekommenderad metod för genereringen av en krypteringsnyckel finns i avsnittet ”Exempel: Generera och använda en krypteringsnyckel.” på sidan 765.

Ansluta till en krypterad databas

Adobe AIR 1.5 och senare

Precis som när du skapar en krypterad databas, ansluter du till krypterade och okrypterade databaser på samma sätt. Den här proceduren beskrivs i detalj i avsnittet ”Ansluta till en databas” på sidan 728. Du använder `open()`-metoden om du vill öppna en anslutning i synkront körningsläge och `openAsync()`-metoden om du vill öppna en anslutning i asynkront körningsläge. Den enda skillnaden när du öppnar en krypterad databas är att du anger korrekt värde för `encryptionKey`-parametern (`open()`-metodens femte parameter och `openAsync()`-metodens sjätte parameter).

Om en felaktig krypteringsnyckel anges returneras ett fel. Om du använder `open()`-metoden genereras ett SQLError-undantag. Om du använder `openAsync()`-metoden skickar SQLConnection-objektet ett SQLErrorEvent-objekt, vars `error`-egenskap innehåller ett SQLError-objekt. I båda fallen har SQLError-objektet som genereras `errorID`-egenskapsvärdet 3138. Fel-ID:t hör till felmeddelandet "Filen som öppnats är inte en databasfil".

Följande exempel illustrerar hur du öppnar en krypterad databas i asynkront körningsläge. För enkelhetens skull är krypteringsnyckeln hårdkodad i programkoden i det här exemplet. Den här tekniken rekommenderas emellertid inte eftersom den inte är säker.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    if (event.error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", event.error.message);
        trace("Details:", event.error.details);
    }
}
```

Följande exempel illustrerar hur du öppnar en krypterad databas i synkront körningsläge. För enkelhetens skull är krypteringsnyckeln hårdkodad i programkoden i det här exemplet. Den här tekniken rekommenderas emellertid inte eftersom den inte är säker.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, SQLMode.UPDATE, false, 1024, encryptionKey);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    if (error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", error.message);
        trace("Details:", error.details);
    }
}
```

Ett exempel som illustrerar en rekommenderad metod för genereringen av en krypteringsnyckel finns i avsnittet ”Exempel: Generera och använda en krypteringsnyckel.” på sidan 765.

Ändra krypteringsnyckeln för en databas

Adobe AIR 1.5 och senare

Om en databas är krypterad kan du ändra krypteringsnyckeln för databasen vid ett senare tillfälle. Om du vill ändra databasens krypteringsnyckel öppna du först en anslutning till databasen genom att skapa en SQLConnection-instans och anropa dess `open()`- eller `openAsync()`-metod. När databasanslutningen har upprättats anropar du `reencrypt()`-metoden och anger den nya krypteringsnyckeln som ett argument.

Som med de flesta databasåtgärder varierar `reencrypt()`-metodens beteende beroende på om synkront eller asynkront körningsläge används för databasanslutningen. Om du använder `open()`-metoden för att ansluta till databasen körs `reencrypt()`-åtgärden synkront. När åtgärden har slutförts exekveras nästa rad i koden:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

Om du i stället upprättar databasanslutningen med `openAsync()`-metoden körs `reencrypt()`-åtgärden asynkront. Omkrypteringen startar när `reencrypt()` anropas. När åtgärden slutförts skickar SQLConnection-objektet en `reencrypt`-händelse. Du använder en händelseavlyssnare för att kontrollera när omkrypteringen är färdig:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event:SQLEvent):void
{
    // save the fact that the key changed
}
```

`reencrypt()`-åtgärden körs i en egen transaktion. Om åtgärden avbryts eller misslyckas (till exempel om programmet stängs innan åtgärden slutförts) återställs transaktionen. I så fall används den ursprungliga krypteringsnyckeln för databasen.

`reencrypt()`-metoden kan inte användas för att ta bort krypteringen från en databas. Om ett `null`-värde eller en krypteringsnyckel som inte är en bytearray på 16 byte skickas till `reencrypt()`-metoden returneras ett fel.

Att tänka på när du använder kryptering med en databas

Adobe AIR 1.5 och senare

I avsnittet ”Användningsområden för en krypterad databas” på sidan 760 illustreras olika situationer då du bör använda en krypterad databas. Självklart har användningsscenarierna för olika program (även dessa och andra scenarier) olika sekretesskrav. Hur du strukturerar användningen av eventuell kryptering i programmet avgör i hög grad hur privat informationen i en databas är. Om du till exempel använder en krypterad databas för att skydda personliga data, även från andra användare på samma dator, behöver varje användares databas en egen krypteringsnyckel. Högst säkerhet erhålls om programmet genererar nyckeln från ett lösenord som anges av användaren. Om krypteringsnyckeln baseras på ett lösenord är informationen inte åtkomlig även om någon lyckas personifiera användarens konto på datorn. Andra gånger kanske du tvärtom vill att en databasfil ska kunna läsas av alla som använder programmet, men inte av andra program. I så fall behöver alla installerade kopior av programmet åtkomst till en delad krypteringsnyckel.

Du kan utforma programmet, och teknikerna som används för att generera krypteringsnyckeln, i enlighet med den sekretessnivå som krävs för programinformationen. Följande lista innehåller designförslag för olika sekretessnivåer:

- Om du vill att en databas ska vara tillgänglig för alla användare som har åtkomst till programmet på alla datorer, använder du en enda nyckel som är tillgänglig för alla programinstanser. Första gången ett program körs kan den delade krypteringsnyckeln hämtas från en server med hjälp av ett säkert protokoll, t.ex. SSL. Därefter kan nyckeln sparas på den krypterade lokala lagringsplatsen för framtida användning. Ett annat alternativ är att kryptera informationen för de enskilda användarna på datorn och synkronisera informationen med ett dataarkiv på en fjärrdator, t.ex. en server, så att informationen blir portabel.
- Om du vill att databasen ska vara tillgänglig för en enskild användare på alla datorer genererar du krypteringsnyckeln från en användarhemlighet, t.ex. ett lösenord. Generera inte nyckeln baserat på ett värde som är kopplat till en särskild dator (t.ex. ett värde som lagras på den krypterade lokala lagringsplatsen). Ett annat alternativ är att kryptera informationen för de enskilda användarna på datorn och synkronisera informationen med ett dataarkiv på en fjärrdator, t.ex. en server, så att informationen blir portabel.
- Om du vill att en databas bara ska vara tillgänglig för en särskild användare på en enskild dator genererar du nyckeln från ett lösenord och ett genererat salt-värde. Ett exempel på den här tekniken finns i avsnittet ”Exempel: Generera och använda en krypteringsnyckel.” på sidan 765.

Nedan följer ytterligare säkerhetsaspekter som är viktiga att ha i åtanke när du utformar ett program som ska använda en krypterad databas:

- Ett system är bara så säkert som dess svagaste länk. Om krypteringsnyckeln genereras baserat på ett användardefinierat lösenord bör du definiera krav på komplexitet och minsta tillåtna längd för lösenorden. Ett kort lösenord som bara innehåller vanliga tecken är inte svårt att lista ut.
- Källkoden för ett AIR-program lagras på en användares dator som oformaterad text (för HTML-innehåll) eller som en binärfil som enkelt kan dekompileras (för SWF-innehåll). Eftersom källkoden är åtkomlig bör du tänka på följande två saker:
  - Hårdkoda aldrig en krypteringsnyckel i källkoden.
  - Förutsätt alltid att tekniken som används för att generera en krypteringsnyckel (t.ex. program som genererar slumpmässiga tecken eller en särskild hash-algoritm) inte innebär några större svårigheter för en angripare.
- Databaskrypteringen i AIR baseras på AES (Advanced Encryption Standard) med CCM-läge (Counter with CBC-MAC). För att det här krypteringschiffret ska vara säkert krävs en användardefinierad nyckel som kombineras med ett salt-värde. Ett exempel finns i avsnittet ”Exempel: Generera och använda en krypteringsnyckel.” på sidan 765.
- När du väljer att kryptera en databas krypteras alla hårddiskfiler som används av databasmotorn i kombination med den databasen. För att förbättra läs- och skrivtiderna i transaktioner sparas emellertid en del data tillfälligt i ett cacheminne. Informationen i cacheminnet är inte krypterad. Om en angripare lyckas få åtkomst till minnet som används av ett AIR-program, t.ex. med hjälp av en felsökare, kan han eller hon komma åt informationen i en öppen databas som inte är krypterad.

Exempel: Generera och använda en krypteringsnyckel.

Adobe AIR 1.5 och senare

Det här exempelprogrammet demonstrerar en teknik som du kan använda för att generera en krypteringsnyckel. Med det här programmet används en hög sekretess- och säkerhetsnivå för att skydda användarnas data. En viktig aspekt vid säkring av privat information är kravet att användaren måste ange ett lösenord varje gång programmet ansluter till databasen. Därför, vilket visas i det här exemplet, ska ett program med den här sekretessnivån aldrig direkt lagra krypteringsnyckeln för databasen.

Programmet består av två delar: en ActionScript-klass som genererar en krypteringsnyckel (klassen EncryptionKeyGenerator), och ett grundläggande användargränssnitt som visar hur den klassen används. Den fullständiga källkoden finns i ”Fullständig exempelkod för generering och användning av en krypteringsnyckel” på sidan 767.

Använda klassen EncryptionKeyGenerator för att få en säker krypteringsnyckel

Adobe AIR 1.5 och senare

Det är inte nödvändigt att förstå exakt hur EncryptionKeyGenerator-klassen fungerar för att kunna använda den i programmet. Mer detaljerad information om hur klassen genererar en krypteringsnyckel för databasen finns i ”EncryptionKeyGenerator-klassen” på sidan 772.

Följ de här stegen om du vill använda EncryptionKeyGenerator-klassen i ditt program:

1. Hämta EncryptionKeyGenerator-klassen som en källkod eller en kompilerad SWC. Klassen EncryptionKeyGenerator ingår i AS3CoreLib-projektet med öppen källkod. Du kan hämta tas3corelib-paketet inklusive källkod och dokumentation. Du kan även hämta SWC-filerna och källkodsfilerna från projektsidan.

2 Lägg källkoden för EncryptionKeyGenerator-klassen (eller as3corelib-SWC) på en plats som programkällkoden kan hitta.

3 Lägg till en `import`-sats i programkällkoden för EncryptionKeyGenerator-klassen.

```
import com.adobe.air.crypto.EncryptionKeyGenerator;
```

4 Före koden som skapar databasen eller en anslutning till den lägger du till kod för att skapa en EncryptionKeyGenerator-instans genom att anropa konstruktorn `EncryptionKeyGenerator()`.

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
```

5 Hämta ett lösenord från användaren:

```
var password:String = passwordInput.text;

if (!keyGenerator.validateStrongPassword(password))
{
    // display an error message
    return;
}
```

EncryptionKeyGenerator-instansen använder det här lösenordet som grund för krypteringsnyckeln (visas i nästa steg). EncryptionKeyGenerator-instansen testar lösenordet mot vissa valideringskrav för starka lösenord. Om valideringen misslyckas inträffar ett fel. Som exempelkoden visar kan du kontrollera lösenordet i förväg genom att anropa EncryptionKeyGenerator-objektets `validateStrongPassword()`-metod. På det sättet kan du avgöra om lösenordet motsvarar minimikraven för starka lösenord och undvika ett fel.

6 Generera krypteringsnyckeln från lösenordet:

```
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

Metoden `getEncryptionKey()` genererar och returnerar krypteringsnyckeln (en 16-byte ByteArray). Sedan kan du använda krypteringsnyckeln för att skapa en ny krypterad databas eller öppna en befintlig.

Lösenordet som hämtas i steg 5 är en obligatorisk parameter i metoden `getEncryptionKey()`.

Obs! *För att få högsta säkerhets- och sekretessnivå för data måste ett program begära att användaren anger ett lösenord varje gång som programmet ansluter till databasen. Lagra inte användarens lösenord eller databasens krypteringsnyckel direkt. Det medför säkerhetsrisker att göra så. I stället, vilket visas i det här exemplet, bör ett program använda samma teknik för att skapa krypteringsnyckeln från lösenordet både när den krypterade databasen skapas och vid senare anslutningar till den.*

Metoden `getEncryptionKey()` accepterar även en andra (valfri) parameter, parametern `overrideSaltELSKey`. EncryptionKeyGenerator skapar ett slumpvärde (kallat *salt*) som används som en del av krypteringsnyckeln. För att kunna återskapa krypteringsnyckeln lagras salt-värdet i ELS-lagret (Encrypted Local Store) i ditt AIR-program. Som standard använder EncryptionKeyGenerator-klassen en speciellt String-värde som ELS-nyckel. Även om det är osannolikt finns en risk för att nyckeln råkar i konflikt med en annan nyckel som används i ditt program. I stället för att använda standardnyckeln kan du ange en egen ELS-nyckel. Om du gör så anger du en egen nyckel genom att skicka den som den andra `getEncryptionKey()`-parametern, vilket visas här:

```
var customKey:String = "My custom ELS salt key";
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password, customKey);
```

7 Skapa eller öppna databasen

Med en krypteringsnyckel som returneras av metoden `getEncryptionKey()` kan din kod skapa en ny krypterad databas eller försöka att öppna den befintliga krypterade databasen. Oavsett vilket kan du använda SQLConnection-klassens `open()`- eller `openAsync()`-metod, vilket beskrivs i ”Skapa en krypterad databas” på sidan 761 och ”Ansluta till en krypterad databas” på sidan 761.

I det här exemplet har programmet utformats att öppna databasen i asynkront körningsläge. Koden aktiverar lämpliga händelseavlyssnare, anropar `openAsync()`-metoden och anger krypteringsnyckeln som det sista argumentet:

```
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
```

Händelseavlyssnarregistreringarna tas bort i avlyssnarmetoderna. Därefter visas ett statusmeddelande som anger om databasen har skapats, öppnats eller om ett fel har inträffat. Den mest intressanta delen i dessa händelsehanterare finns i `openError()`-metoden. I den metoden kontrollerar en `if`-sats om databasen finns (d.v.s. att koden försöker ansluta till en befintlig databas) och om fel-id:t matchar konstanten `EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID`. Om båda dessa villkor har värdet true betyder det sannolikt att lösenordet som användaren angav är felaktigt. (Det kan även betyda att den angivna filen inte är en databasfil.) Följande kod kontrollerar fel-id:t:

```
if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.text = "Incorrect password!";
}
else
{
    statusMsg.text = "Error creating or opening database.";
}
```

Den fullständiga koden för exemplets händelseavlyssnare finns i ”Fullständig exempelkod för generering och användning av en krypteringsnyckel” på sidan 767.

Fullständig exempelkod för generering och användning av en krypteringsnyckel

Adobe AIR 1.5 och senare

Här nedan är den fullständiga koden för exempelprogrammet "Generera och använda en krypteringsnyckel." Koden består av två delar.

I det här exemplet används klassen EncryptionKeyGenerator för att skapa en krypteringsnyckel från ett lösenord. Klassen EncryptionKeyGenerator ingår i AS3CoreLib-projektet med öppen källkod. Du kan hämta tas3corelib-paketet inklusive källkod och dokumentation. Du kan även hämta SWC-filerna och källkodsfilerna från projektsidan.

Flex-exempel

MXML-filen i programmet innehåller källkoden för ett enkelt program som skapar eller öppnar en anslutning till en krypterad databas:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted
database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must
contain at least one lowercase letter, at least one uppercase letter, and at least one number
or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);
```

```
                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next
time you open the application, you will need to re-enter the password to open the database
again." width="75%" height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

Flash Professional-exempel

FLA-filen i programmet innehåller källkoden för ett enkelt program som skapar eller öppnar en anslutning till en krypterad databas. FLA-filen har fyra komponenter som är placerade på scenen:

Instansnamn	Komponenttyp	Beskrivning
`instruktioner`	Label	Innehåller de instruktioner som lämnats till användaren
`passwordInput`	TextInput	Inmatningsfält där användaren anger lösenordet
`openButton`	Button	Knapp som användaren klickar på efter att lösenordet har angetts
`statusMsg`	Label	Visar statusmeddelanden (fel eller godkänt)

Programmets kod är angiven på en nyckelbildruta på ruta 1 på huvudtidslinjen. Programmets kod följer nedan:

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time
you open the application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
    {
        statusMsg.text = "Please specify a password.";
        return;
```

```
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);

    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

EncryptionKeyGenerator-klassen

Adobe AIR 1.5 och senare

Det är inte nödvändigt att förstå EncryptionKeyGenerator-klassen i detalj för att använda den till att skapa en säker krypteringsnyckel till din programdatabas. Sättet du använder klassen på förklaras i "Använda klassen EncryptionKeyGenerator för att få en säker krypteringsnyckel" på sidan 765. Det kan dock vara värdefullt att förstå teknikerna som klassen använder. Du kanske vill anpassa klassen eller införliva vissa av dess tekniker för tillfällen då en annan sekretessnivå är nödvändig.

Klassen EncryptionKeyGenerator ingår i AS3CoreLib-projektet med öppen källkod. Du kan hämta as3corelib-paketet inklusive källkod och dokumentation.Du kan även se källkoden på projektwebbplatsen eller ladda ned den för att arbeta med vägledning av förklaringarna.

När koden skapar en EncryptionKeyGenerator-instans och anropar dess `getEncryptionKey()`-metod vidtas flera steg för att säkerställa att bara rätt användare kan komma åt data. Processen är samma vid generering av en krypteringsnyckel från ett användarangivet lösenord innan databasen skapas, samt vid återskapande av krypteringsnyckeln för att öppna databasen.

Begära och validera ett starkt lösenord

Adobe AIR 1.5 och senare

När koden anropar `getEncryptionKey()`-metoden skickas ett lösenord som en parameter. Lösenordet används som bas för krypteringsnyckeln. Genom att använda information som bara användaren känner till försäkrar den här designen att endast användare som har tillgång till lösenordet kan komma åt data i databasen. Även om en angripare lyckas komma åt användarens konto på datorn måste han eller hon känna till lösenordet för att få åtkomst till databasen. Som en extra säkerhetsåtgärd lagras lösenordet aldrig av programmet.

Koden för programmet skapar en EncryptionKeyGenerator-instans och anropar metoden `getEncryptionKey()`, vilket skickar det användarangivna lösenordet som ett argument (variabeln `password` i det här exemplet):

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

Första steget som EncryptionKeyGenerator-klassen tar när `getEncryptionKey()`-metoden anropas är att kontrollera det användarangivna lösenordet för att se till att det motsvarar kraven på starka lösenord. EncryptionKeyGenerator-klassen kräver att ett lösenord är 8–32 tecken långt. Lösenordet måste innehålla både versaler och gemener och minst en siffra eller symbol.

Det reguljära uttrycket som kontrollerar det här mönstret är definierat som en konstant som heter `STRONG_PASSWORD_PATTERN`:

```
private static const STRONG_PASSWORD_PATTERN:RegExp =
/(?=^.{8,32}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

Koden som kontrollerar lösenordet finns i EncryptionKeyGenerator-klassens `validateStrongPassword()`-metod. Koden är som följer:

```
public function vaidateStrongPassword(password:String):Boolean
{
    if (password == null || password.length <= 0)
    {
        return false;
    }

    return STRONG_PASSWORD_PATTERN.test(password))
}
```

Metoden `getEncryptionKey()` anropar internt EncryptionKeyGenerator-klassens `validateStrongPassword()`-metod och genererar ett undantag om lösenordet är ogiltigt. Metoden `validateStrongPassword()` är en offentlig metod så att programkod kan kontrollera ett lösenord utan att anropa `getEncryptionKey()`-metoden för att undvika att ett fel inträffar.

Utöka lösenordet till 256 bitar

Adobe AIR 1.5 och senare

Längre fram i processen finns instruktioner som anger att lösenordet måste vara 256 bitar långt. I stället för att tvinga varje användare att ange ett lösenord som är exakt 256 bitar (32 tecken) skapar koden ett längre lösenord genom att upprepa tecknen i lösenordet.

Metoden `getEncryptionKey()` anropar `concatenatePassword()`-metoden för att den ska skapa det långa lösenordet.

```
var concatenatedPassword:String = concatenatePassword(password);
```

Följande kod gäller för `concatenatePassword()`-metoden:

```
private function concatenatePassword(pwd:String):String
{
    var len:int = pwd.length;
    var targetLength:int = 32;

    if (len == targetLength)
    {
        return pwd;
    }

    var repetitions:int = Math.floor(targetLength / len);
    var excess:int = targetLength % len;

    var result:String = "";

    for (var i:uint = 0; i < repetitions; i++)
    {
        result += pwd;
    }

    result += pwd.substr(0, excess);

    return result;
}
```

Om lösenordet är kortare än 256 bitar drar koden samman lösenordet (konkatenerar) så att det blir 256 bitar. Om längden inte blir exakt 256 bitar kortas den sista konkateneringen av så att lösenordet blir exakt 256 bitar.

Generera eller hämta ett 256-bitars salt-värde

Adobe AIR 1.5 och senare

Nästa steg är att få fram ett 256-bitars salt-värde som i ett senare steg kombineras med lösenordet. Ett *salt*-värde är ett slumpmässigt värde som läggs till eller kombineras med ett användardefinierat värde för att generera ett lösenord. Även om en användare väljer ett vanligt ord som lösenord garanterar "lösenord och salt-värde"-kombinationen att ett slumpmässigt värde används av systemet. Denna slumpmässighet hjälper till att skydda mot en ordlisteattack, där en angripare använder en lista med ord för att hitta ett lösenord. Genom att generera salt-värdet och lagra det på den krypterade lokala lagringsplatsen kopplas det dessutom till användarens konto på datorn där databasfilen finns.

Om programmet anropar `getEncryptionKey()`-metoden för första gången skapar koden ett slumpmässigt 256-bitars salt-värde. Annars läser koden in salt-värdet från den krypterade lokala lagringsplatsen.

Salt-värdet lagras i en variabel som heter `salt`. Koden avgör om ett salt-värde redan har skapats genom att försöka att läsa in salt-värdet från den krypterade lokala lagringsplatsen:

```
var salt:ByteArray = EncryptedLocalStore.getItem(saltKey);
if (salt == null)
{
    salt = makeSalt();
    EncryptedLocalStore.setItem(saltKey, salt);
}
```

Om koden skapar ett nytt salt-värde genererar `makeSalt()`-metoden ett slumpmässigt 256-bitars värde. Eftersom värdet lagras på den krypterade lokala lagringsplatsen genereras det som ett ByteArray-objekt. `makeSalt()`-metoden använder `Math.random()`-metoden för att generera det slumpmässiga värdet. `Math.random()`-metoden kan inte generera 256 bitar på en gång. I stället används en slinga som anropar `Math.random()` åtta gånger. Varje gång genereras ett slumpmässigt uint-värde mellan 0 och 4294967295 (det högsta uint-värdet). Ett uint-värde används av praktiska skäl eftersom det innehåller exakt 32 bitar. Ett 256-bitars värde genereras genom att åtta uint-värden skrivs till bytearrayen. Här följer koden för `makeSalt()`-metoden:

```
private function makeSalt():ByteArray
{
    var result:ByteArray = new ByteArray;

    for (var i:uint = 0; i < 8; i++)
    {
        result.writeUnsignedInt(Math.round(Math.random() * uint.MAX_VALUE));
    }

    return result;
}
```

När koden sparar salt-värdet till den krypterade lokala lagringsplatsen (ELS) eller hämtar salt-värdet från ELS, behövs en String-nyckel som salt-värdet sparas under. Salt-värdet kan inte hämtas om nyckeln är okänd. Om det är fallet kan inte krypteringsnyckeln återskapas varje gång för att öppna om databasen. Som standard använder EncryptionKeyGenerator en fördefinierad ELS-nyckel som definieras i konstanten `SALT_ELS_KEY`. I stället för att använda standardnyckeln kan programkoden ange en ELS-nyckel som används i anropet till `getEncryptionKey()`-metoden. Standardnyckeln eller den programanpassade salt-ELS-nyckeln lagras i en variabel som heter `saltKey`. Variabeln används i anropen till `EncryptedLocalStore.setItem()` och `EncryptedLocalStore.getItem()`, som visades tidigare.

Kombinera 256-bitars lösenordet och salt-värdet med hjälp av XOR-operatorn

Adobe AIR 1.5 och senare

Koden har nu ett 256-bitars lösenord och ett 256-bitars salt-värde. Sedan använder det en XOR-åtgärd i bitform för att kombinera salt-värdet och det konkatenerade lösenordet till ett enda värde. Den här tekniken skapar ett 256-bitars lösenord som består av tecken från hela spektrat av möjliga tecken. Den här regeln har värdet true även om det faktiska lösenordet mest sannolikt består av alfanumeriska tecken till största delen. Den här ökade slumpmässigheten har fördelen att uppsättningen möjliga lösenord blir stor utan att användaren måste ange ett långt och komplext lösenord.

Resultatet av XOR-åtgärden lagras i variabeln `unhashedKey`. Själva processen med att kombinera de båda värdena med hjälp av en XOR-åtgärd sker i `xorBytes()`-metoden:

```
var unhashedKey:ByteArray = xorBytes(concatenatedPassword, salt);
```

Den bitvisa XOR-operatorn (`^`) tar två uint-värden och returnerar ett uint-värde. (Ett uint-värde består av 32 bitar.) Indatavärdena som skickas som argument till `xorBytes()`-metoden är en sträng (lösenordet) och en bytearray (salt-värdet). En slinga används för att extrahera 32 bitar från varje indatavärde som ska kombineras med hjälp av XOR-operatorn.

```
private function xorBytes(passwordString:String, salt:ByteArray):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 4)
    {
        // ...
    }

    return result;
}
```

Inom loopen extraheras de första 32 bitarna (4 byte) från `passwordString`-parametern. Dessa bitar extraheras och konverteras till ett uint-värde (`o1`) i en tvåstegsprocess. Först hämtar `charCodeAt()`-metoden varje teckens numeriska värde. Därefter flyttas värdet till rätt position i uint med hjälp av operatorn för flyttning till vänster (`<<`) och det flyttade värdet läggs till `o1`. Det första tecknet (`i`) blir exempelvis de första 8 bitarna genom att bitarna flyttas 24 bitar åt vänster med hjälp av operatorn för flyttning till vänster (`<<`) och genom att värdet sedan associeras med `o1`. Det andra tecknet `(i + 1)` blir de efterföljande 8 bitarna genom att tillhörande värde flyttas 16 bitar åt vänster och genom att resultatet sedan läggs till `o1`. Värdena för det tredje och fjärde tecknet läggs till på samma sätt.

```
        // ...

        // Extract 4 bytes from the password string and convert to a uint
        var o1:uint = passwordString.charCodeAt(i) << 24;
        o1 += passwordString.charCodeAt(i + 1) << 16;
        o1 += passwordString.charCodeAt(i + 2) << 8;
        o1 += passwordString.charCodeAt(i + 3);

        // ...
```

Variabeln `o1` innehåller nu 32 bitar från parametern `passwordString`. Därefter extraheras 32 bitar från parametern `salt` genom att dess `readUnsignedInt()`-metod anropas. De 32 bitarna lagras i uint-variabeln `o2`.

```
        // ...

        salt.position = i;
        var o2:uint = salt.readUnsignedInt();

        // ...
```

Slutligen kombineras de två 32-bitarsvärdena (uint) med XOR-operatorn och resultatet skrivs till en ByteArray som heter `result`.

```
        // ...

        var xor:uint = o1 ^ o2;
        result.writeUnsignedInt(xor);
        // ...
```

När slingan slutförts returneras den bytearray som innehåller XOR-resultatet.

```
        // ...
    }

    return result;
}
```

Hash-koda nyckeln

Adobe AIR 1.5 och senare

När det konkatenerade lösenordet och salt-värdet har kombinerats är nästa steg att ytterligare säkra det här värdet genom att hash-koda det med hash-algoritmen SHA-256. Hash-kodningen försvårar reverse engineering av en potentiell angripare.

I det här skedet har koden en ByteArray som heter `unhashedKey` och innehåller det konkatenerade lösenordet kombinerat med salt-värdet. Core-biblioteksprojektet för ActionScript 3.0 (as3corelib) innehåller en SHA256-klass i paketet com.adobe.crypto. Metoden `SHA256.hashBytes()` utför en SHA-256-hash-kodning på en ByteArray och returnerar ett String-värde med hash-kodningsresultatet på 256 bitar som ett hexadecimalt tal. Klassen EncryptionKeyGenerator använder SHA256-klassen för att hash-koda nyckeln:

```
var hashedKey:String = SHA256.hashBytes(unhashedKey);
```

Extrahera krypteringsnyckeln från hashen

Adobe AIR 1.5 och senare

Krypteringsnyckeln måste vara en bytearray som är exakt 16 byte (128 bitar) lång. Resultatet av SHA-256-hash-algoritmen är alltid 256 bitar långt. Följaktligen är det sista steget att välja 128 bitar från det hash-kodade resultatet som ska användas som den faktiska krypteringsnyckeln.

I EncryptionKeyGenerator-klassen reducerar koden nyckeln till 128 bitar genom att anropa `generateEncryptionKey()`-metoden. Sedan returneras metodens resultat som resultatet för `getEncryptionKey()`-metoden:

```
var encryptionKey:ByteArray = generateEncryptionKey(hashedKey);
return encryptionKey;
```

Du måste inte använda de första 128 bitarna som krypteringsnyckeln. Du kan välja från ett urval bitar med godtycklig startplats, du kan välja alla bitar eller använda ett annat sätt att välja bitar. Det viktiga är att 128 olika bitar väljs, och att samma 128 bitar används varje gång.

I det här fallet använder `generateEncryptionKey()`-metoden urvalet av bitar med start vid 18:e byte som krypteringsnyckeln. Som vi nämnt tidigare returnerar SHA256-klassen en sträng som innehåller en 256-bitars hash i form av ett hexadecimalt tal. Ett enskilt block med 128 bitar har för många byte för att alla ska kunna läggas till i en bytearray på en gång. Av den anledningen används en `for`-slinga för att extrahera tecken från den hexadecimala strängen, konvertera dem till numeriska värden och lägga till dem i bytearrayen. Det resulterande String-värdet för SHA-256 är 64 tecken långt. Ett intervall med 128 bitar är lika med 32 tecken i String-värdet och varje tecken representerar 4 bitar. Den minsta dataökningen som du kan lägga till i en bytearray är en byte (8 bitar), vilket motsvarar två tecken i `hash`-strängen. Slingan räknar följaktligen från 0 till 31 (32 tecken) i steg om 2 tecken.

I slingan börjar koden med att fastställa startpunkten för det aktuella teckenparet. Eftersom vi vill börja intervallet vid tecknet på indexposition 17 (den 18:e byten) tilldelas `position`-variabeln det aktuella iteratorvärdet (`i`) plus 17. De två tecknen på den aktuella positionen extraheras med hjälp av strängobjektets `substr()`-metod. Tecknen lagras i variabeln `hex`. Därefter används `parseInt()`-metoden för att konvertera `hex`-strängen till ett decimalt heltal. Det lagrar det värdet i int-variabeln `byte`. Till sist lägger koden till värdet i `byte` till ByteArrayen `result` genom att använda `writeByte()`-metoden. När slingan slutförts innehåller bytearrayen `result` 16 byte och kan användas som krypteringsnyckel för en databas.

```
private function generateEncryptionKey(hash:String):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 2)
    {
        var position:uint = i + 17;
        var hex:String = hash.substr(position, 2);
        var byte:int = parseInt(hex, 16);
        result.writeByte(byte);
    }

    return result;
}
```

Strategier för att arbeta med SQL-databaser

Adobe AIR 1.0 och senare

Ett program kan få åtkomst till och arbeta med en lokal SQL-databas på flera olika sätt. Programdesignen kan variera när det gäller hur programkoden är organiserad, i vilken ordning och när åtgärder utförs o.s.v. Det kan bli enklare att utveckla programmet beroende på vilka tekniker du använder. Med rätt tekniker blir programmet också enklare att ändra i kommande uppdateringar. Dessutom påverkar teknikerna hur bra programmet fungerar ur användarnas perspektiv.

Distribuera en förifylld databas

Adobe AIR 1.0 och senare

När du använder en lokal SQL-databas (AIR) i ditt program förväntas att du använder en databas med en viss struktur för tabeller, kolumner o.s.v. För vissa program måste även vissa data vara ifyllda i databasfilen. Ett sätt att säkerställa att databasen har rätt struktur är att skapa databasen i programkoden. När programmet läses in görs en kontroll om dess databasfil finns på en viss plats. Om filen inte finns körs en rad kommandon för att skapa en databasfil, skapa en databasstruktur och fylla i tabeller med initiala data.

Koden som skapar databasen och dess tabeller är ofta komplex. Ofta används den bara en gång under tiden programmet är installerat, men den påverkar kontinuerligt programmets storlek och komplexitet. Som ett alternativ till att skapa databasen, strukturen och data med programmering, kan du distribuera en förifylld databas med ditt program. När du distribuerar en fördefinierad databas inkluderar du databasfilen i programmets AIR-paket.

Liksom alla filer som inkluderas i ett AIR-paket, installeras en paketerad databasfil i programkatalogen (katalogen representeras av egenskapen `File.applicationDirectory`). Filerna i den katalogen är dock skrivskyddade. Använd filen i AIR-paketet som mall för databasen. Första gången en användare kör programmet ska användaren kopiera den ursprungliga databasfilen till sin ”Peka mot programlagringskatalogen” på sidan 674 (eller någon annan plats) och använda den databasen i programmet.

Bästa teknikerna för att arbeta med lokala SQL-databaser

Adobe AIR 1.0 och senare

Nedan ges förslag på olika tekniker som du kan använda för att underlätta hanteringen av programmen och förbättra programmens prestanda och säkerhet när du arbetar med lokala SQL-databaser.

Skapa databasanslutningar i förväg

Adobe AIR 1.0 och senare

Även om inga satser körs i programmet vid första inläsningen kan du undvika fördröjningar när satserna körs genom att initiera ett SQLConnection-objekt och anropa dess `open()`- eller `openAsync()`-metod i förväg (till exempel efter programstarten). Se ”Ansluta till en databas” på sidan 728.

Återanvänd databasanslutningar

Adobe AIR 1.0 och senare

Om du ansluter till en viss databas under programmets körningstid kan du spara en referens till SQLConnection-instansen och återanvända den i programmet, i stället för att stänga och öppna anslutningen på nytt. Se ”Ansluta till en databas” på sidan 728.

Prioritera asynkront körningsläge

Adobe AIR 1.0 och senare

När du skriver kod för dataåtkomst kan det vara lockande att köra åtgärderna synkront i stället för asynkront, eftersom synkrona åtgärder ofta kräver kortare och mindre komplicerad kod. Synkrona åtgärder kan dock påverka prestanda på ett märkbart sätt för användarna och försämra deras upplevelse av programmet (se avsnittet ”Använda synkrona och asynkrona databasåtgärder” på sidan 755). Tiden det tar att utföra en åtgärd varierar beroende på åtgärden och inte minst på mängden data som ingår i åtgärden. En `INSERT` -sats (SQL) som bara lägger till en rad i databasen tar till exempel mindre tid än en `SELECT`-sats som hämtar tusentals rader data. När du använder synkron körning för att utföra flera åtgärder är dock åtgärderna vanligtvis sammankopplade. Även om det tar mycket kort tid att utföra varje enskild åtgärd fryses programmet tills alla synkrona åtgärder har slutförts. Den sammanlagda tiden för alla sammankopplade åtgärder kan vara tillräcklig för att stoppa programmet.

Använd asynkrona åtgärder som standard, särskilt om ett stort antal rader ingår i åtgärderna. Det finns en teknik för att dela upp bearbetningen i stora mängder `SELECT`-satsresultat (se avsnittet ”Hämta SELECT-resultat i delar” på sidan 744). Den här tekniken kan dock endast användas i asynkront körningsläge. Använd synkrona åtgärder endast när du inte kan få önskat resultat med asynkron programmering, när du har övervägt inverkan på prestanda som programmets användare får erfara och när du har testat programmet så att du vet hur programmets prestanda påverkas. Du måste kanske skriva en mer komplicerad kod när du använder asynkron körning. Tänk på att du endast behöver skriva koden en gång, men att programmets användare måste använda den upprepade gånger, oavsett om den tar kort eller lång tid att köra.

Om du använder en separat SQLStatement-instans för varje SQL-sats kan flera SQL-åtgärder placeras i kö samtidigt. Den asynkrona koden skrivs i så fall på liknande sätt som den synkrona koden. Mer information finns i ”Om asynkront körningsläge” på sidan 758.

Använd separata SQL-satser och ändra inte SQLStatement-instansens textegenskap

Adobe AIR 1.0 och senare

Skapa en separat SQLStatement-instans för varje SQL-sats som körs fler än en gång i programmet. Använd den SQLStatement-instansen varje gång SQL-kommandot körs. Anta till exempel att du skapar ett program som innehåller fyra olika SQL-åtgärder som utförs flera gånger. I så fall skapar du fyra separata SQLStatement-instanser och kör varje sats genom att anropa dess `execute()`-metod. Undvik att använda en enskild SQLStatement-instans för alla SQL-satser och omdefiniera `text`-egenskapen före varje körning av satsen.

Använd satsparametrar

Adobe AIR 1.0 och senare

Använd SQLStatement-parametrar. Sammanfoga aldrig användarens indata med satstexten. Programmet blir säkrare om du använder parametrar eftersom det skyddar mot SQL-injektionsangrepp. Du kan dessutom använda objekt i frågor (i stället för endast SQL-litteralvärden). Satserna körs även mer effektivt eftersom de kan återanvändas utan att de behöver kompileras om varje gång de körs. Mer information finns i ”Använda parametrar i programsatser” på sidan 732.

Kapitel 41: Arbeta med bytearrayer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen ByteArray kan du läsa från och skriva till en binär ström med data, som är en array med byte. Den här klassen gör att du kan få åtkomst till data på den mest grundläggande nivån. Eftersom data består av byte, eller grupper om åtta bitar, innebär förmågan att läsa data i byte att du kan komma åt data för vilka det inte finns några klasser eller åtkomstmetoder. Med klassen ByteArray kan du analysera alla dataströmmar, från en bitmapp till en ström med data som färdas över ett nätverk, på bytenivå.

Med metoden `writeObject()` kan du skriva ett objekt i AMF-serialisering (Action Message Format) till en ByteArray, och med metoden `readObject()` kan du läsa ett serialiserat objekt från en ByteArray till en variabel med den ursprungliga datatypen. Du kan serialisera alla objekt utom visningsobjekt, det vill säga objekt som kan placeras i visningslistan. Du kan också tilldela serialiserade objekt tillbaka till anpassade klassinstanser om den anpassade klassen är tillgänglig under körning. När du har konverterat ett objekt till AMF kan du överföra det via en nätverksanslutning eller spara det i en fil.

Exemplet på Adobe® AIR®-program som beskrivs här läser en ZIP-fil som ett exempel på hur en byteström bearbetas, extraherar en lista med de filer som finns i ZIP-filen och skriver dem till skrivbordet.

Fler hjälpavsnitt

flash.utils.ByteArray

flash.utils.IExternalizable

AMF-specifikationer

Läsa och skriva en ByteArray

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen ByteArray tillhör paketet flash.utils. Om du vill skapa ett ByteArray-objekt i ActionScript 3.0 importerar du klassen ByteArray och anropar konstruktorn, vilket visas i exemplet nedan:

```
import flash.utils.ByteArray;
var stream:ByteArray = new ByteArray();
```

ByteArray-metoder

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Alla meningsfulla dataströmmar ordnas i ett format som du kan analysera så att du hittar den information du vill ha. I en post i en enkel anställningsfil tar du till exempel med ID-nummer, namn, adress, telefonnummer och så vidare. En MP3-ljudfil innehåller en ID3-tagg som identifierar titel, författare, album, publiceringsdatum och genre för filen som hämtas. Det här formatet gör att du vet i vilken ordning som data i strömmen förväntas. Du kan läsa byteströmmen på ett effektivt sätt.

Klassen ByteArray omfattar flera metoder som gör det enklare att läsa från och skriva till en dataström. En del av dessa metoder omfattar `readBytes()` och `writeBytes()`, `readInt()` och `writeInt()`, `readFloat()` och `writeFloat()`, `readObject()` och `writeObject()` samt `readUTFBytes()` och `writeUTFBytes()`. Med dessa metoder kan du läsa data från dataströmmen till variabler med särskilda datatyper och skriva från särskilda datatyper direkt i den binära dataströmmen.

Följande kod läser en enkel array med strängar och flyttal och skriver respektive element till en ByteArray. Strukturen på arrayen gör det möjligt för koden att anropa lämpliga ByteArray-metoder (`writeUTFBytes()` och `writeFloat()`) för att skriva data. Det återkommande datamönstret gör det möjligt att läsa arrayen med en slinga.

```
// The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

import flash.utils.ByteArray;

// define the grocery list Array
var groceries:Array = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes:ByteArray = new ByteArray();
// for each item in the array
for (var i:int = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        trace("bytes.position is: " + bytes.position);//display the position in ByteArray
}
trace("bytes length is: " +  bytes.length);// display the length
```

Egenskapen position

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskapen position lagrar den aktuella positionen för pekaren som indexerar ByteArray vid läsning eller skrivning. Startvärdet för egenskapen position är 0 (noll) vilket visas i koden nedan:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
```

När du läser från eller skriver till en ByteArray uppdaterar metoder som du använder egenskapen position till att peka på platsen omedelbart efter den sista byte som lästes eller skrevs. I följande kod skrivs en sträng till en ByteArray och sedan pekar egenskapen position mot den byte som finns omedelbart efter strängen i ByteArray:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
```

En läsåtgärd ökar på samma sätt egenskapen position med det antal byte som läses.

```
var bytes:ByteArray = new ByteArray();

trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

Observera att du kan ange egenskapen position till en plats i en ByteArray om du vill läsa eller skriva vid den förskjutningen.

Egenskaperna bytesAvailable och length

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskaperna `length` och `bytesAvailable` anger hur lång en ByteArray är och hur många byte som finns kvar i den från den aktuella positionen till slutet. I följande exempel visas hur du kan använda dessa egenskaper. I exemplet skrivs en sträng med text till en ByteArray och sedan läses denna ByteArray med en byte i taget tills tecknet ”a” eller slutet (`bytesAvailable <= 0`) påträffas.

```
var bytes:ByteArray = new ByteArray();
var text:String = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    trace("Found the letter a; position is: " + bytes.position); // 23
    trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

Egenskapen endian

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Olika datorer kan lagra multibytetal på olika sätt, vilket innebär att tal kanske kräver mer än en byte med minne för att lagras. Ett heltal kanske till exempel behöver fyra byte, eller 32 bitar, med minne. Vissa datorer lagrar den mest signifikanta byten i siffran först, i den lägsta minnesadressen, och andra lagrar den minst signifikanta byten först. Denna byteordning kallas för *big endian* (mest signifikanta byte först) eller *little endian* (minst signifikanta byte först). Talet 0x31323334 skulle till exempel lagras på följande sätt med byteordningen big endian respektive little endian, där a0 representerar den lägsta minnesdressen på fyra byte och a3 representerar den högsta:

Big Endian	**Big Endian**	**Big Endian**	**Big Endian**
a0	a1	a2	a3
31	32	33	34

Little Endian	**Little Endian**	**Little Endian**	**Little Endian**
a0	a1	a2	a3
34	33	32	31

Med egenskapen `endian` för klassen ByteArray kan du ange denna byteordning för multibytetal som du bearbetar. Godkända värden för den här egenskapen är `"bigEndian"` och `"littleEndian"` och klassen Endian definierar konstanterna `BIG_ENDIAN` och `LITTLE_ENDIAN` när egenskapen `endian` ställs in med dessa strängar.

Metoderna compress() och uncompress()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `compress()` kan du komprimera en ByteArray enligt en komprimeringsalgoritm som du anger som en parameter. Med metoden `uncompress()` kan du dekomprimera en komprimerad ByteArray enligt en komprimeringsalgoritm. När du har anropat `compress()` och `uncompress()` ställs längden på bytearrayen in på den nya längden och egenskapen position ställs in på slutet.

Klassen CompressionAlgorithm definierar vilka konstanter som du kan använda för att ange komprimeringsalgoritmen. Klassen ByteArray har stöd för algoritmerna deflate (endast AIR), zlib och lzma. Komprimeringsformatet zlib beskrivs på http://www.ietf.org/rfc/rfc1950.txt. lzma-algoritmen lades till i Flash Player 11.4 och AIR 3.4. Den beskrivs på http://www.7-zip.org/7z.html.

Dekomprimeringsalgoritmen används i flera komprimeringsformat, till exempel zlib, gzip och en del zip-implementationer. Deflate-komprimeringsalgoritmen beskrivs på http://www.ietf.org/rfc/rfc1951.txt.

I följande exempel komprimeras en ByteArray med namnet `bytes` som använder lzma-algoritmen:

```
bytes.compress(CompressionAlgorithm.LZMA);
```

I följande exempel dekomprimeras en komprimerad ByteArray som använder dekomprimeringsalgoritmen:

```
bytes.uncompress(CompressionAlgorithm.LZMA);
```

Läsa och skriva objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoderna `readObject()` och `writeObject()` kan läsa ett objekt från och skriva ett objekt till en ByteArray, kodad med AMF-serialisering (Action Message Format). AMF är ett tillverkarspecifikt meddelandeprotokoll som har skapats av Adobe och används av olika ActionScript 3.0-klasser, bland annat Netstream, NetConnection, NetStream, LocalConnection och Shared Objects.

En typmarkör på en byte beskriver vilken typ av kodade data som följer. AMF använder följande 13 datatyper:

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

Den kodade datan kommer efter typmarkören såvida inte markören representerar ett enda möjligt värde, till exempel null eller sant eller falskt. Då kodas inget annat.

Det finns två versioner av AMF: AMF0 och AMF3. AMF 0 har stöd för att skicka komplexa objekt via referens och tillåter slutpunkter att återställa objektrelationer. AMF 3 förbättrar AMF 0 genom att skicka objekt-traits och strängar via referens, förutom objektreferenserna, och genom att ha stöd för de nya datatyper som introducerades i ActionScript 3.0. Egenskapen `ByteArray.objectEcoding` anger vilken version av AMF som används för att koda objektdata. Klassen flash.net.ObjectEncoding definierar konstanter som anger AMF-versionen: `ObjectEncoding.AMF0` och `ObjectEncoding.AMF3`.

Följande exempel anropar `writeObject()` för att skriva ett XML-objekt till en ByteArray. Sedan komprimeras objektet med hjälp av dekomprimeringsalgoritmen och skrivs till filen `order` på skrivbordet. I exemplet används en etikett för att visa meddelandet ”Wrote order file to desktop!” i AIR-fönstret när det är klart.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;
public class WriteObjectExample extends Sprite
{
    public function WriteObjectExample()
    {
        var bytes:ByteArray = new ByteArray();
        var myLabel:TextField = new TextField();
        myLabel.x = 150;
        myLabel.y = 150;
        myLabel.width = 200;
        addChild(myLabel);

        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>;

        // Write XML object to ByteArray
        bytes.writeObject(myXML);
        bytes.position = 0;//reset position to beginning
        bytes.compress(CompressionAlgorithm.DEFLATE);// compress ByteArray
        writeBytesToFile("order.xml", bytes);
        myLabel.text = "Wrote order file to desktop!";
    }

    private function writeBytesToFile(fileName:String, data:ByteArray):void
    {
        var outFile:File = File.desktopDirectory; // dest folder is desktop
        outFile = outFile.resolvePath(fileName);  // name of file to write
        var outStream:FileStream = new FileStream();
        // open output file stream in WRITE mode
        outStream.open(outFile, FileMode.WRITE);
        // write out the file
        outStream.writeBytes(data, 0, data.length);
        // close it
        outStream.close();
    }
}
```

Metoden `readObject()` läser ett objekt i serialiserat AMF från en ByteArray och lagrar det i ett objekt av angiven typ. I följande exempel läser filen `order` från skrivbordet till en ByteArray (`inBytes`), dekomprimerar den och anropar `readObject()` för att lagra den i XML-objektet `orderXML`. I exemplet används en `for each()`-slingakonstruktor för att lägga till varje nod i ett textområde så att de visas. I exemplet visas också värdet på egenskapen `objectEncoding` tillsammans med huvudet för innehållet i filen `order`.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;

public class ReadObjectExample extends Sprite
{
    public function ReadObjectExample()
    {
        var inBytes:ByteArray = new ByteArray();
        // define text area for displaying XML content
        var myTxt:TextField = new TextField();
        myTxt.width = 550;
        myTxt.height = 400;
        addChild(myTxt);
        //display objectEncoding and file heading
        myTxt.text = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";
        readFileIntoByteArray("order", inBytes);

        inBytes.position = 0; // reset position to beginning
        inBytes.uncompress(CompressionAlgorithm.DEFLATE);
        inBytes.position = 0;//reset position to beginning
        // read XML Object
        var orderXML:XML = inBytes.readObject();

        // for each node in orderXML
        for each (var child:XML in orderXML)
        {
            // append child node to text area
            myTxt.text += child + "\n";
        }
    }

    // read specified file into byte array
    private function readFileIntoByteArray(fileName:String, data:ByteArray):void
    {
        var inFile:File = File.desktopDirectory; // source folder is desktop
        inFile = inFile.resolvePath(fileName);  // name of file to read
        var inStream:FileStream = new FileStream();
        inStream.open(inFile, FileMode.READ);
        inStream.readBytes(data);
        inStream.close();
    }
}
```

Exempel på ByteArray: Läsa en ZIP-fil

Adobe AIR 1.0 och senare

I det här exemplet visas hur en enkel ZIP-fil som innehåller flera filer av olika typ läses. Relevanta data extraheras från metadata för respektive fil, varje fil dekomprimeras till en ByteArray och filen skrivs till skrivbordet.

Den allmänna strukturen för en ZIP-fil baseras på specifikationen från PKWARE Inc., som beskrivs på http://www.pkware.com/documents/casestudies/APPNOTE.TXT. Inledningsvis finns ett filhuvud och fildata för den första filen i ZIP-arkivet, följt av ett filhuvud- och fildatapar för varje ytterligare fil. (Filhuvudets struktur beskrivs senare.) Sedan kan ZIP-filen omfatta en databeskrivningspost (vanligtvis när utdata-ZIP-filen skapas i minnet i stället för att sparas på en disk). Därefter finns det flera valfria element: arkivets dekrypteringshuvud, arkivets extra datapost, central katalogstruktur, Zip64-slutpost för central katalog, Zip64-slutlokaliserare för central katalog och slutpost för central katalog.

Koden i det här exemplet skrivs bara för att analysera ZIP-filer som inte innehåller mappar och den förväntar inga databeskrivningsposter. Den ignorerar all information efter den sista fildatan.

Respektive filhuvud har följande format:

filhuvudets signatur	4 byte
obligatorisk version	2 byte
allmän bitflagga	2 byte
komprimeringsmetod	2 byte (8=DEKOMPRIMERA; 0=OKOMPRIMERAD)
tid för senast ändrade fil	2 byte
datum för senast ändrade fil	2 byte
crc-32	4 byte
komprimerad storlek	4 byte
okomprimerad storlek	4 byte
längd på filnamn	2 byte
längd på extra fält	2 byte
filnamn	variabel
extra fält	variabel

Efter filhuvudet finns den faktiska fildatan, som kan vara komprimerad eller okomprimerad, beroende på komprimeringsmetodflaggan. Flaggan är 0 (noll) om fildatan är okomprimerad och 8 om den har komprimerats med algoritmen DEFLATE eller ett annat värde för andra komprimeringsalgoritmer.

Användargränssnittet för det här exemplet består av en etikett och ett textområde (`taFiles`). Programmet skriver följande information i textområdet för varje fil som påträffas i ZIP-filen: filnamnet, komprimeringsstorleken och den okomprimerade storleken. Följande MXML-dokument definierar användargränssnittet för Flex-versionen av programmet:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

I början av programmet utförs följande uppgifter:

- De nödvändiga klasserna importeras

```
import flash.filesystem.*;
import flash.utils.ByteArray;
import flash.events.Event;
```

- Definierar användargränssnittet för Flash

```
import fl.controls.*;

//requires TextArea and Label components in the Library
var taFiles = new TextArea();
var output = new Label();
taFiles.setSize(320, 150);
taFiles.move(10, 30);
output.move(10, 10);
output.width = 150;
output.text = "Contents of HelloAir.zip";
addChild(taFiles);
addChild(output);
```

- ByteArrayen `bytes` definieras

```
var bytes:ByteArray = new ByteArray();
```

- Variabler som lagrar metadata från filhuvudet definieras

```
// variables for reading fixed portion of file header
var fileName:String = new String();
var flNameLength:uint;
var xfldLength:uint;
var offset:uint;
var compSize:uint;
var uncompSize:uint;
var compMethod:int;
var signature:int;
```

- Objekten File (`zfile`) och FileStream (`zStream`) definieras till att representera ZIP-filen, och platsen för ZIP-filen som filerna extraheras från definieras – en fil med namnet ”HelloAIR.zip” i skrivbordskatalogen.

```
// File variables for accessing .zip file
var zfile:File = File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream:FileStream = new FileStream();
```

I Flex startar programkoden i metoden `init()`, som anropas som `creationComplete`-hanterare för rottaggen `mx:WindowedApplication`.

```
// for Flex
private function init():void
{
```

Programmet börjar med att öppna ZIP-filen i READ-läge.

```
    zStream.open(zfile, FileMode.READ);
```

Sedan ställer det in egenskapen `endian` för `bytes` på `LITTLE_ENDIAN` vilket anger att byteordningen för numeriska fält har den minst signifikanta byten först.

```
    bytes.endian = Endian.LITTLE_ENDIAN;
```

Sedan börjar en `while()`-programsats en slinga som fortsätter tills den aktuella placeringen i filströmmen är större än eller lika med storleken på filen.

```
    while (zStream.position < zfile.size)
    {
```

Den första programsatsen i slingan läser de första 30 byten i filströmmen till ByteArrayen `bytes`. Dessa första 30 byte utgör delen med fast storlek i det första filhuvudet.

```
        // read fixed metadata portion of local file header
        zStream.readBytes(bytes, 0, 30);
```

Sedan läser koden ett heltal (`signature`) från några av de första byten i huvudet om 30 byte. ZIP-formatdefinitionen anger att signaturen för varje filhuvud är det hexadecimala värdet `0x04034b50`. Om signaturen är en annan innebär det att koden har flyttats utanför fildelen av ZIP-filen och det finns inga fler filer att extrahera. I så fall avslutar koden `while`-slingan omedelbart i stället för att vänta till slutet på bytearrayen.

```
        bytes.position = 0;
        signature = bytes.readInt();
        // if no longer reading data files, quit
        if (signature != 0x04034b50)
        {
            break;
        }
```

Nästa del av koden läser huvudbyten vid förskjutningsposition 8 och lagrar värdet i variabeln `compMethod`. Denna byte innehåller ett värde som anger vilken komprimeringsmetod som användes för att komprimera filen. Det går att använda olika komprimeringsmetoder, men i praktiken använder nästan alla ZIP-filer komprimeringsalgoritmen DEFLATE. Om den aktuella filen har komprimerats med DEFLATE-komprimering är `compMethod` lika med 8. Om filen är okomprimerad är `compMethod` lika med 0.

```
        bytes.position = 8;
        compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

Efter dessa första 30 byte finns en del av huvudet med variabel längd som innehåller filnamn och ett extra fält om det är möjligt. Storleken på den här delen lagras i variabeln `offset`. Storleken beräknas genom att längden på filnamnet adderas till längden på det extra fältet, som läses från huvudet vid förskjutningen 26 och 28.

```
offset = 0;// stores length of variable portion of metadata
bytes.position = 26;  // offset to file name length
flNameLength = bytes.readShort();// store file name
offset += flNameLength; // add length of file name
bytes.position = 28;// offset to extra field length
xfldLength = bytes.readShort();
offset += xfldLength;// add length of extra field
```

Sedan läser programmet delen av filhuvudet med variabel längd och söker efter antalet byte som lagras i variabeln `offset`.

```
// read variable length bytes between fixed-length header and compressed file data
zStream.readBytes(bytes, 30, offset);
```

Programmet läser filnamnet från den variabla delen av huvudet och visar det i textområdet tillsammans med den komprimerade och okomprimerade (ursprungliga) storleken på filen.

```
// Flash version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.appendText(fileName + "\n"); // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.appendText("\tCompressed size is: " + compSize + '\n');
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.appendText("\tUncompressed size is: " + uncompSize + '\n');


// Flex version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.text += fileName + "\n"; // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.text += "\tCompressed size is: " + compSize + '\n';
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.text += "\tUncompressed size is: " + uncompSize + '\n';
```

Exemplet läser resten av filen från filströmmen till `bytes` och söker efter längden som anges av komprimeringsstorleken, och skriver över filhuvudet i de första 30 byten. Den komprimerade storleken är korrekt även om filen inte är komprimerad, eftersom den komprimerade storleken i detta fall är lika med den okomprimerade storleken på filen.

```
// read compressed file to offset 0 of bytes; for uncompressed files
// the compressed and uncompressed size is the same
if (compSize == 0) continue;
zStream.readBytes(bytes, 0, compSize);
```

Sedan dekomprimerar exemplet den komprimerade filen och anropar funktionen `outfile()` för att skriva den till utdatafilströmmen. Det skickar filnamnet och bytearrayen som innehåller fildata till `outfile()`.

```
if (compMethod == 8) // if file is compressed, uncompress
{
    bytes.uncompress(CompressionAlgorithm.DEFLATE);
}
outFile(fileName, bytes);   // call outFile() to write out the file
```

I det tidigare nämnda exemplet kommer `bytes.uncompress(CompressionAlgorithm.DEFLATE)` endast att fungera i AIR-program. För att få komprimerade data okomprimerade för både AIR och Flash Player, anropar du `inflate()`-funktionen i ByteArray.

De avslutande klammerparenteserna anger slutet på `while`-slingan, och för `init()`-metoden och Flex-programkoden, med undantag för metoden `outFile()`. Körningen gör en slinga tillbaka till början av `while`-slingan och fortsätter att bearbeta nästa byte i ZIP-filen, antingen med att extrahera en annan fil eller avsluta bearbetningen av ZIP-filen om den sista filen har bearbetats.

```
    } // end of while loop
} // for Flex version, end of init() method and application
```

Funktionen `outfile()` öppnar en utdatafil i WRITE-läge på skrivbordet, och ger den namnet som anges i parametern `filename`. Sedan skrivs fildata från `data`-parametern till utdatafilströmmen (`outStream`) och filen stängs.

```
// Flash version
function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // destination folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}

private function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

Kapitel 42: Grundläggande om nätverk och kommunikation

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar program i Flash Player eller AIR behöver du ofta ha tillgång till resurser utanför programmet. Du kanske till exempel vill skicka en begäran för en bild på en webbserver på Internet och få tillbaka bildens data. Eller så kanske du vill skicka serialiserade objekt fram och tillbaka via en socketanslutning med en programserver. API:erna i Flash Player och AIR innehåller klasser som gör det möjligt för dina program att delta i den här överföringen. Dessa API:er stöder IP-baserade nätverk för protokoll som UDP, TCP, HTTP, RTMP och RTMFP.

Följande klasser kan användas för att skicka och ta emot data över ett nätverk:

Klass	Dataformat som stöds	Protokoll	Beskrivning
Loader	SWF, PNG, JPEG, GIF	HTTP, HTTPS	Läser in datatyper som stöds och konverterar dessa data till ett visningsobjekt. Se ”Dynamisk inläsning av visningsinnehåll” på sidan 198.
URLLoader	Alla (text, XML, binära filer o.s.v.)	HTTP, HTTPS	Läser in arbiträra dataformat. Programmet ansvarar för tolkningen av data. Se ”Använda klassen URLLoader” på sidan 817
FileReference	Alla	HTTP	Överföra och hämta filer. Se ”Använda klassen FileReference” på sidan 655
NetConnection	Video, ljud, ActionScript Message Format (AMF)	HTTP, HTTPS, RTMP, RTMFP	Ansluter till video-, ljud- och fjärrobjektströmmar. Se ”Arbeta med video” på sidan 474.
Sound	Ljud	HTTP	Läser in och spelar upp ljudformat som stöds. Se ”Läsa in externa ljudfiler” på sidan 443.
XMLSocket	XML	TCP	Överför XML-meddelanden med en XMLSocket-server. Se ”XML-socketar” på sidan 804.
Socket	Alla	TCP	Ansluter till en TCP-socketserver. Se ”Binära klientsocketar” på sidan 800.
SecureSocket (AIR)	Alla	TCP med SSLv3 eller TLSv1	Ansluter till en TCP-socketserver som kräver SSL- eller TLS-säkerhet. Se ”Säkra klientsocketar (AIR)” på sidan 800.
ServerSocket (AIR)	Alla	TCP	Fungerar som en server för inkommande TCP-socketanslutningar. Se ”Serversocketar” på sidan 808.
DatagramSocket (AIR)	Alla	UDP	Skickar och tar emot UDP-paket. Se ”UDP-socketar (AIR)” på sidan 810

När du skapar ett webbprogram kan det ofta vara användbart att kontinuerligt lagra information om användarens programläge. HTML-sidor och program använder vanligtvis cookies för det. I Flash Player kan klassen SharedObject användas för detta. Se ”Delade objekt” på sidan 703. (Klassen SharedObject kan användas i AIR-program, men det är färre begränsningar när data helt enkelt sparas till en vanlig fil.)

När ditt Flash Player- eller AIR-program behöver kommunicera med ett annat Flash Player- eller AIR-program på samma dator kan du använda klassen LocalConnection. Till exempel kan två (eller flera) SWF-filer på samma webbsida kommunicera sinsemellan. På samma sätt kan en SWF-fil som körs på en webbsida kommunicera med ett AIR-program. Se ”Kommunicera med andra Flash Player- och AIR-instanser” på sidan 832.

Om du behöver kommunicera med annat än SWF-filer på den lokala datorn kan du använda klassen NativeProcess i AIR 2. Med hjälp av klassen NativeProcess kan ditt AIR-program starta och kommunicera med andra program. Se ”Kommunicera med interna processer i AIR” på sidan 839.

Om du behöver information om nätverksmiljön för den dator på vilken ett AIR-program körs kan du använda följande klasser:

- NetworkInfo – Ger information om de tillgängliga nätverksgränssnitten, till exempel datorns IP-adress. Se ”Nätverksgränssnitt” på sidan 793.
- DNSResolver – Gör att du kan söka efter DNS-poster. Se ”DNS-poster (Domain Name System)” på sidan 796.
- ServiceMonitor – Låter dig övervaka tillgängligheten för en server. Se ”Tjänstövervakning” på sidan 794.
- URLMonitor – Låter dig övervaka tillgängligheten för en resurs på en viss URL. Se ”HTTP-övervakning” på sidan 795.
- SocketMonitor och SecureSocketMonitor – Låter dig övervaka tillgängligheten för en resurs vid en socket. Se ”Socketövervakning” på sidan 796.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer som du kommer att stöta på när du programmerar nätverks- och kommunikationskod:

Externa data Data som lagras i någon form utanför programmet och som läses in till programmet vid behov. Dessa data kan lagras i en fil som läses in direkt, eller också kan de lagras i en databas eller i någon annan form som hämtas genom att anropa skript eller program som körs på en server.

URL-kodade variabler Med det URL-kodade formatet kan flera variabler (par av variabelnamn och -värden) representeras i en enskild textsträng. De enskilda variablerna skrivs i formatet namn=värde. Varje variabel (d.v.s. varje par av namn–värde) avgränsas med et-tecken på följande sätt: variabel1=värde1&variabel2=värde2. På så sätt kan ett oändligt antal variabler skickas som ett enda meddelande.

MIME-typ En standardkod som används för att identifiera typen för en viss fil vid Internet-kommunikation. Varje filtyp har en särskild kod som används för att identifiera den. När en fil eller ett meddelande skickas, kommer en dator (som en webbserver eller en användares Flash Player- eller AIR-instans) att ange vilken typ av fil som skickas.

HTTP Hypertext Transfer Protocol är ett standardformat för att skicka webbsidor och andra typer av innehåll på Internet.

Metod för begärande När ett program (till exempel ett AIR-program eller en webbläsare) skickar ett meddelande (som kallas för HTTP-begäran) till en webbserver, kan de data som skickas vara inbäddade i begäran på ett av två olika sätt. Dessa båda sätt är de två förfrågningsmetoderna GET och POST. På servern kommer det program som tar emot begäran att behöva undersöka lämplig del av begäran för att hitta data. Därför måste den förfrågningsmetod som används för att skicka data från programmet överensstämma med den förfrågningsmetod som används för att läsa data på servern.

Socketanslutning En beständig anslutning för kommunikation mellan två datorer.

Överför Skicka en fil till en annan dator.

Hämta Hämta en fil från en annan dator.

Nätverksgränssnitt

Adobe AIR 2 och senare

Du kan använda NetworkInfo-objektet för att identifiera de nätverksgränssnitt för hårdvara och mjukvara som finns tillgängliga för programmet. NetworkInfo-objektet är ett *singleton*-objekt, du behöver inte skapa något. Använd i stället den statiska klassegenskapen `networkInfo` om du vill komma åt det enskilda NetworkInfo-objektet. NetworkInfo-objektet skickar också en `networkChange`-händelse när något av de tillgängliga gränssnitten ändras.

Anropa metoden `findInterfaces()` för att hämta en lista med NetworkInterface-objekt. Varje NetworkInterface-objekt i listan beskriver ett av de tillgängliga gränssnitten. NetworkInterface-objektet tillhandahåller sådan information som IP-adress, maskinvaruadress, största överföringsenhet och om gränssnittet är aktivt eller inte.

I följande kodexempel spåras NetworkInterface-egenskaper för alla gränssnitt på klientdatorn:

```
package {
import flash.display.Sprite;
import flash.net.InterfaceAddress;
import flash.net.NetworkInfo;
import flash.net.NetworkInterface;

public class NetworkInformationExample extends Sprite
{
    public function NetworkInformationExample()
    {
        var networkInfo:NetworkInfo = NetworkInfo.networkInfo;
        var interfaces:Vector.<NetworkInterface> = networkInfo.findInterfaces();

        if( interfaces != null )
        {
            trace( "Interface count: " + interfaces.length );
            for each ( var interfaceObj:NetworkInterface in interfaces )
            {
                trace( "\nname: " + interfaceObj.name );
                trace( "display name: " + interfaceObj.displayName );
                trace( "mtu: "  + interfaceObj.mtu );
                trace( "active?: " + interfaceObj.active );
                trace( "parent interface: " + interfaceObj.parent );
                trace( "hardware address: " + interfaceObj.hardwareAddress );
                if( interfaceObj.subInterfaces != null )
                {
                    trace( "# subinterfaces: " + interfaceObj.subInterfaces.length );
                }
                trace("# addresses: " + interfaceObj.addresses.length );
                for each ( var address:InterfaceAddress in interfaceObj.addresses )
                {
                    trace( "  type: "           + address.ipVersion );
                    trace( "  address: " + address.address );
                    trace( "  broadcast: " + address.broadcast );
                    trace( "  prefix length: " + address.prefixLength );
                }
            }
        }
    }
}
}
```

Mer information finns i:

- NetworkInfo
- NetworkInterface
- InterfaceAddress
- Flexpert: Detecting the network connection type with Flex 4.5 (Upptäcka nätverksanslutningstyp med Flex)

Ändringar i nätverksanslutning

Adobe AIR 1.0 och senare

AIR-programmet kan köras i miljöer där nätverksanslutningen är osäker och ändras. Om en nätverksanslutning blir tillgänglig eller inte längre kan användas, skickar Adobe AIR en nätverksändringshändelse. Det blir då lättare för ett program att hantera anslutningarna till olika onlineresurser. Både NetworkInfo-objektet och programmets NativeApplication-objekt skickar händelsen `networkChange`. Om programmet ska reagera på den här händelsen lägger du till en lyssnare:

```
NetworkInfo.networkInfo.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

Ange också en händelsehanterarfunktion:

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

Händelsen `networkChange` indikerar inte någon ändring i all nätverksaktivitet, utan bara att en enskild nätverksanslutning har ändrats. AIR försöker inte tolka innebörden av den här nätverksändringen. En nätverksansluten dator kan ha många verkliga och virtuella anslutningar. Att en anslutning försvinner behöver alltså inte innebära att en resurs går förlorad. Nya anslutningar garanterar inte heller förbättrad resurstillgänglighet. Ibland kan en ny anslutning till och med blockera åtkomst till resurser som varit tillgängliga tidigare (till exempel vid anslutning till VPN).

Vanligtvis kan ett program bara avgöra om det kan ansluta till en fjärresurs genom att prova. I ramverket för tjänstövervakning tillhandahålls en händelsebaserad metod att svara på ändringar i nätverksanslutning till en angiven värd.

Obs! *Ramverket för tjänstövervakning upptäcker om en server svarar godkännande på en begäran. Att kontrollen slutförs garanterar inte fullständig anslutning. Skalbara webbtjänster använder ofta program för cachelagring och belastningsutjämning för att dirigera om trafik till ett kluster med webbservrar. I en sådan situation kan tjänsteleverantörerna bara tillhandahålla en ofullständig diagnos för nätverksanslutningen.*

Tjänstövervakning

Adobe AIR 1.0 och senare

Tjänstövervakningens ramverk, som är skilt från AIR-ramverket, finns i filen aircore.swc. Om det här ramverket ska kunna användas måste filen aircore.swc tas med i ditt programbygge.

I Adobe® Flash® Builder inkluderas det här biblioteket automatiskt.

Klassen ServiceMonitor implementerar ramverket för övervakning av nätverkstjänster och ger en basfunktion för tjänstövervakning. Som standard skickar en instans av klassen ServiceMonitor händelser avseende nätverksanslutning. ServiceMonitor-objektet skickar dessa händelser när instansen skapas och när körningstiden upptäcker en nätverksändring. Du kan dessutom ställa in egenskapen `pollInterval` i en ServiceMonitor-instans så att anslutningen kontrolleras vid ett angivet intervall i millisekunder, oavsett vanliga nätverksanslutningshändelser. Ett ServiceMonitor-objekt kontrollerar inte nätverksanslutningen förrän metoden `start()` anropas.

Klassen URLMonitor, en underklass till klassen ServiceMonitor, upptäcker ändringar i HTTP-anslutningen för en angiven URLRequest.

Klassen SocketMonitor, som också är en underklass till klassen ServiceMonitor, upptäcker ändringar i anslutningen till en särskild värd vid en särskild port.

Obs! *Före AIR 2 publicerades tjänstövervakningens ramverk i servicemonitor.swc-biblioteket. Detta bibliotek är nu inaktuellt. Du ska använda aircore.swc-biblioteket i stället.*

Flash CS4 och CS5 Professional

Så här använder du dessa klasser i Adobe® Flash® CS4 eller CS5 Professional:

1. Välj kommandot Arkiv > Publiceringsinställningar.
2. Klicka på knappen Inställningar för ActionScript 3.0. Välj Bibliotekssökväg.
3. Klicka på knappen Bläddra efter SWC-fil och bläddra till AIK-mappen i installationsmappen för Flash Professional.
4. I den här mappen ska du leta reda på filen /frameworks/libs/air/aircore.swc (för AIR 2) eller filen /frameworks/libs/air/servicemonitor.swc (för AIR 1.5).
5. Klicka på OK.
6. Lägg till följande importsats till ActionScript 3.0-koden:

```
import air.net.*;
```

Flash CS3 Professional

Om du vill använda dessa klasser i Adobe® Flash® CS3 Professional drar du komponenten ServiceMonitorShim från komponentpanelen till biblioteket och lägger sedan till följande `import`-sats till ActionScript 3.0-koden:

```
import air.net.*;
```

HTTP-övervakning

Adobe AIR 1.0 och senare

Klassen URLMonitor bestämmer om HTTP-begäran ska göras till en angiven adress vid port 80 (den vanliga porten för HTTP-kommunikation). Följande kod använder en instans av klassen URLMonitor för att identifiera anslutningsändringar för Adobes webbplats:

```
import air.net.URLMonitor;
import flash.net.URLRequest;
import flash.events.StatusEvent;
var monitor:URLMonitor;
monitor = new URLMonitor(new URLRequest('http://www.example.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

Socketövervakning

Adobe AIR 1.0 och senare

AIR-program kan också använda socketanslutningar för anslutning av push model-typ. Nätverkskommunikation via otillåtna portar begränsas vanligtvis av brandväggar och nätverksroutrar av säkerhetsskäl. Utvecklarna måste därför tänka på att användarna kanske inte alltid har möjlighet att skapa socketanslutningar.

Följande kod använder en instans av klassen SocketMonitor för att identifiera anslutningsändringar för en socketanslutning. Porten som övervakas är 6667, en gemensam port för IRC:

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.example.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

Om socketservern kräver en säker anslutning kan du använda klassen SecureSocketMonitor istället för SocketMonitor.

DNS-poster (Domain Name System)

Adobe AIR 2.0 och senare

Du kan leta upp DNS-resursposter med hjälp av klassen DNSResolver. I DNS-resursposter tillhandahålls information som IP-adressen för ett domännamn och domännamnet för en IP-adress. Du kan leta upp följande typer av DNS-resursposter:

- ARecord – IPv4-adress för en värd.
- AAAARecord – IPv6-adress för en värd.
- MXRecord – post för utbyte av e-post för en värd.
- PTRRecord – värdnamn för en IP-adress.
- SRVRecord – tjänstpost för en tjänst.

Om du vill leta upp en post skickar du en frågesträng och det klassobjekt som representerar posttypen till `lookup()`-metoden för DNSResolver-objektet. Vilken frågesträng som ska användas beror på posttypen:

Postklass	Frågesträng	Exempel på frågesträng
ARecord	värdnamn	"example.com"
AAAARecord	värdnamn	"example.com"
MXRecord	värdnamn	"example.com"
PTRRecord	IP-adress	"208.77.188.166"
SRVRecord	Tjänstidentifierare: _service._protocol.host	"_sip._tcp.example.com"

I följande kodexempel letas IP-adressen för värden "example.com" upp.

```
package
{
    import flash.display.Sprite;
    import flash.events.DNSResolverEvent;
    import flash.events.ErrorEvent;
    import flash.net.dns.ARecord;
    import flash.net.dns.DNSResolver;

    public class DNSResolverExample extends Sprite
    {

        public function DNSResolverExample()
        {
            var resolver:DNSResolver = new DNSResolver();
            resolver.addEventListener( DNSResolverEvent.LOOKUP, lookupComplete );
            resolver.addEventListener( ErrorEvent.ERROR, lookupError );

            resolver.lookup( "example.com.", ARecord );
        }

        private function lookupComplete( event:DNSResolverEvent ):void
        {
            trace( "Query string: " + event.host );
            trace( "Record count: " + event.resourceRecords.length );
            for each( var record:* in event.resourceRecords )
            {
                if( record is ARecord ) trace( record.address );
            }

        }

        private function lookupError( error:ErrorEvent ):void
        {
            trace("Error: " + error.text );
        }
    }
}
```

Mer information finns i:

- DNSResolver
- DNSResolverEvent
- ARecord

- AAAARecord
- MXRecord
- PTRRecord
- SRVRecord

Kapitel 43: Socketanslutningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En socket är en typ av nätverksanslutning som upprättas mellan två datorprocesser. Vanligtvis körs processerna på två olika datorer anslutna till samma IP-nätverk. Men de anslutna processerna kan köras på samma dator med IP-adressen för den lokala värden.

Adobe Flash Player stöder TCP-socketanslutningar på klientsidan. Ett Flash Player-program kan ansluta till en annan process genom att fungera som socketserver, men kan inte godkänna en inkommande anslutningsbegäran från andra processer. Ett Flash Player-program kan med andra ord ansluta till en TCP-server, men kan inte fungera som en.

Flash Player API:et innehåller också klassen XMLSocket. Klassen XMLSocket använder ett protokoll specifikt för Flash Player som gör det möjligt att utbyta XML-meddelanden med en server som stöder det protokollet. Klassen XMLSocket introducerades i ActionScript 1 och stöds fortfarande för att ge bakåtkompatibilitet. Generellt sett bör Socket-klassen användas för nya program, såvida du inte ansluter till en server som skapats specifikt för att kommunicera med Flash XMLSockets.

Adobe AIR innehåller flera klasser för socketbaserad nätverksprogrammering. AIR-program kan fungera som TCP-socketservrar med klassen ServerSocket och kan ansluta till socketservrar som kräver SSL- eller TLS-säkerhet med klassen SecureSocket. AIR-program kan också skicka och ta emot UDP-meddelanden (Universal Datagram Protocol) med klassen DatagramSocket.

Fler hjälpavsnitt

flash.net-paket

”Ansluta till socketar” på sidan 1073

TCP-socketar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Transmission Control Protocol (TCP) är ett sätt att utbyta meddelanden över en beständig nätverksanslutning. TCP garanterar att skickade meddelanden kommer fram i rätt ordning (bortsett från vid stora nätverksproblem). TCP-anslutningar kräver en "klient" och en "server". Flash Player kan skapa klientsocketar. Adobe AIR kan dessutom skapa serversocketar.

Följande ActionScript-API:er tillhandahåller TCP-anslutningar:

- Socket – gör att ett klientprogram kan ansluta till en server. Socket-klassen kan inte känna av inkommande anslutningar.
- SecureSocket (AIR) – gör att ett klientprogram kan ansluta till en betrodd server och delta i krypterad kommunikation.
- ServerSocket (AIR) – gör att ett program kan känna av inkommande anslutningar och agera som server.
- XMLSocket – gör att ett klientprogram kan ansluta till en XMLSocket-server.

Binära klientsocketar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En binär socketanslutning liknar en XML-socket förutom att klienten och servern inte är begränsade till att utbyta XML-meddelanden. Anslutningen kan i stället överföra data som binär information. Det gör att du kan ansluta till ett brett utbud av tjänster, däribland e-postservrar (POP3, SMTP och IMAP) och nyhetsservrar (NNTP).

Socket, klass

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen Socket kan du skapa socketanslutningar och läsa och skriva binära rådata. Klassen Socket är användbar när du arbetar med servrar som har binära protokoll. Genom att använda binära socketanslutningar kan du skriva kod som möjliggör interaktion med flera olika Internet-protokoll, till exempel POP3, SMTP, IMAP och NNTP. Detta innebär i sin tur att dina program kan anslutas till e-post- och nyhetsservrar.

Flash Player kan samverka med en server genom att använda den serverns binära protokoll direkt. Vissa servrar använder bigendian-byteordningen, medan andra använder littleendian-byteordningen. De flesta servrar på Internet använder bigendian-byteordningen eftersom ”nätverksbyteordningen” är bigendian. Littleendian-byteordningen är populär eftersom den används av Intel® x86-arkitekturen. Du bör använda den endian-byteordning som matchar byteordningen för den server som sänder eller tar emot data. Alla åtgärder som utförs av IDataInput- och IDataOutput-gränssnitten, och klasserna som implementerar de gränssnitten (ByteArray, Socket och URLStream), är som standard kodade i formatet bigendian, d.v.s. med den viktigaste byten först. Denna standardbyteordning har valts för att matcha Java och den officiella nätverksbyteordningen. Om du vill ändra byteordningen som används till bigendian eller littleendian kan du ställa in egenskapen `endian` på `Endian.BIG_ENDIAN` eller `Endian.LITTLE_ENDIAN`.

Klassen Socket ärver alla de metoder som definieras av gränssnitten IDataInput och IDataOutput (finns i paketet flash.utils). De här metoderna måste användas för att skriva till och läsa från socketen.

Mer information finns i:

- Socket
- IDataInput
- IDataOutput
- socketData-händelse

Säkra klientsocketar (AIR)

Adobe AIR 2 och senare

Du kan använda klassen SecureSocket om du vill ansluta till socketservrar som använder Secure Sockets Layer version 4 (SSLv4) eller Transport Layer Security version 1 (TLSv1). Det finns tre fördelar med en säker socket: serverautentisering, dataintegritet och meddelandesekretess. Körningen autentiserar en server med hjälp av servercertifikatet och förhållandet till rotcertifikaten eller de mellanliggande certifikaten från certifikatutfärdare i användarens förtroendearkiv. Körningstiden beror på de krypteringsalgoritmer som används av SSL- och TSL-protokollimplementeringar för att tillhandahålla dataintegritet och meddelandesekretess.

När du ansluter till en server med objektet SecureSocket valideras servercertifikatet av körtiden med hjälp av certifikatets förtroendearkiv. I Windows och Mac tillhandahålls förtroendearkivet av operativsystemet. I Linux tillhandahålls ett eget förtroendearkiv av körningen.

Om inte servercertifikatet är giltigt eller betrott skickas en `ioError`-händelse av körningstiden. Du kan kontrollera egenskapen `serverCertificateStatus` för objektet SecureSocket för att fastställa varför valideringen misslyckades. Det tillhandahålls ingen åtgärd för att kommunicera med en server som inte har något giltigt och betrott certifikat.

Klassen CertificateStatus definierar strängkonstanter som representerar de möjliga valideringsresultaten:

- Expired – utgångsdatumet har passerat.
- Invalid – ett certifikat kan vara ogiltigt av flera olika anledningar. Certifikatet kan till exempel ha ändrats eller skadats, eller vara fel typ av certifikat.
- Invalid chain – ett eller flera certifikat i serverns certifikatkedja är ogiltiga.
- Principal mismatch – serverns värdnamn och certifikatets namn överensstämmer inte. Med andra ord används fel certifikat av servern.
- Revoked – certifikatutfärdaren har återkallat certifikatet.
- Trusted – certifikatet är giltigt och betrott. Ett SecureSocket-objekt kan endast anslutas till en server där ett giltigt, betrott certifikat används.
- Unknown – SecureSocket-objektet har inte validerat certifikatet än. Egenskapen `serverCertificateStatus` har det här statusvärdet innan du anropar `connect()` och innan någon av händelserna `connect` och `ioError` skickas.
- Untrusted signers – det går inte att koppla certifikatet till något betrott rotcertifikat i klientdatorns förtroendearkiv.

För kommunikation med ett SecureSocket-objekt krävs en server som använder ett säkert protokoll och har ett giltigt, betrott certifikat. För övrigt är det samma sak att använda ett SecureSocket-objekt som att använda ett Socket-objekt.

Det finns inte stöd för SecureSocket-objektet på alla plattformar. Använd egenskapen `isSupported` i klassen SecureSocket för att testa om det finns stöd i körtiden för att använda SecureSocket-objektet på den aktuella klientdatorn.

Mer information finns i:

- SecureSocket
- CertificateStatus
- IDataInput
- IDataOutput
- socketData-händelse

Exempel på TCP-socket: skapa en Telnet-klient

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Telnet-exemplet visar tekniker som kan användas för att ansluta till en fjärrserver och överföra data med hjälp av klassen Socket. Exemplet demonstrerar följande tekniker:

- Skapa en anpassad Telnet-klient med hjälp av klassen Socket
- Skicka text till fjärrservern med hjälp av ett ByteArray-objekt
- Hantera mottagna data från en fjärrserver

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Telnet-programfilerna finns i mappen Samples/Telnet. Programmet består av följande filer:

Fil	Beskrivning
TelnetSocket.fla eller TelnetSocket.mxml	Huvudprogramfilen som består av användargränssnittet för Flex (MXML) eller Flash (FLA).
TelnetSocket.as	Dokumentklass som tillhandahåller användargränssnittslogiken (endast Flash).
com/example/programmingas3/Telnet/Telnet.as	Erbjuder Telnet-klientfunktionaliteten för programmet, som att ansluta till en fjärrserver och skicka, ta emot och visa data.

Översikt över Telnet-socketprogrammet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Huvudfilen TelnetSocket.mxml ansvarar för att skapa användargränssnittet (UI) för hela programmet.

Utöver användargränssnittet definierar den här filen även två metoder, `login()` och `sendCommand()`, för att ansluta användaren till den angivna servern.

Följande kod anger ActionScript i huvudprogramfilen:

```
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 = "";
}
```

Den första raden kod importerar klassen Telnet från det anpassade paketet com.example.programmingas.socket. Den andra raden kod deklarerar en instans av klassen Telnet, `telnetClient`, som kommer att initieras senare av metoden `connect()`. Sedan deklareras metoden `connect()` och initierar den tidigare deklarerade `telnetClient`-variabeln. Den här metoden skickar det av användaren angivna Telnet-servernamnet, Telnet-serverporten och en referens till en TextArea-komponent i visningslistan, som används för att visa textsvaren från socketservern. De sista två raderna för metoden `connect()` ställer in egenskapen `title` för panelen och aktiverar Panel-komponenten, som gör att användaren kan skicka data till fjärrservern. Den sista metoden i huvudprogramfilen, `sendCommand()`, används för att skicka användarens kommandon till fjärrservern som ett ByteArray-objekt.

Översikt över klassen Telnet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Telnet ansvarar för att ansluta till Telnet-fjärrservern och för att skicka/ta emot data.

Klassen Telnet deklarerar följande privata variabler:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

Den första variabeln, `serverURL`, innehåller den av användaren angivna serveradressen som anslutningen berör.

Den andra variabeln, `portNumber`, är portnumret som Telnet-servern för närvarande körs på. Standardinställningen innebär att Telnet-tjänsten körs på port 23.

Den tredje variabeln, `socket`, är en Socket-instans som kommer att försöka att ansluta till servern som har definierats av variablerna `serverURL` och `portNumber`.

Den fjärde variabeln, `ta`, är en referens till en TextArea-komponentinstans på scenen. Den här komponenten används för att visa svar från Telnet-fjärrservern och eventuella felmeddelanden.

Den sista variabeln, `state`, är ett numeriskt värde som används för att avgöra vilka alternativ din Telnet-klient stöder.

Som du redan har sett anropas klassen Telnets konstruktorfunktion av metoden `connect()` i huvudprogramfilen.

Telnet-konstruktorn har tre parametrar: `server`, `port` och `output`. Parametrarna `server` och `port` anger servernamnet och portnumret där Telnet-servern körs. Den sista parametern, `output`, är en referens till en TextArea-komponentinstans på scenen där serverutdata kommer att visas för användarna.

```
public function Telnet(server:String, port:int, output: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();
    }
}
```

Skriva data till en socket

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill skriva data till en socketanslutning anropar du någon av skrivmetoderna i klassen Socket. De här skrivmetoderna omfattar `writeBoolean()`, `writeByte()`, `writeBytes()` och `writeDouble()` med flera. Sedan tömmer du dina data i utdatabufferten med hjälp av metoden `flush()`. På Telnet-servern skrivs data till socketanslutningen med hjälp av metoden `writeBytes()`, som tar bytearrayen som parameter och skickar den till utdatabufferten. Koden för metoden `writeBytesToSocket()` är följande:

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Den här metoden anropas av metoden `sendCommand()` i huvudprogramfilen.

Visa meddelanden från socketservern

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Varje gång ett meddelande tas emot från socketservern, eller när en händelse inträffar, anropas den anpassade metoden `msg()`. Den här metoden bifogar en sträng till TextArea på scenen och anropar en anpassad `setScroll()`-metod, vilket gör att TextArea-komponenten rullar så att den hamnar allra längst ned. Koden för metoden `msg()` är följande:

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Om du inte rullade innehållet i TextArea-komponenten automatiskt måste användarna dra rullningslisterna i textområdet manuellt för att se det senaste svaret från servern.

Rulla en TextArea-komponent

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `setScroll()` innehåller en enda ActionScript-rad som rullar TextArea-komponentens innehåll lodrätt så att användaren kan se den sista raden med returnerad text. Följande utdrag visar metoden `setScroll()`:

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Den här metoden ställer in egenskapen `verticalScrollPosition`, som är radnumret för den översta raden med tecken som visas, och ställer in värdet för egenskapen `maxVerticalScrollPosition`.

XML-socketar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med en XML-socket kan du skapa en anslutning till en fjärrserver som förblir öppen tills den uttryckligen stängs. Du kan utbyta strängdata, till exempel XML, mellan servern och klienten. En annan fördel med att använda en XML-socketserver är att klienten inte uttryckligen behöver begära data. Servern kan skicka data utan att vänta på en begäran och skicka data till alla anslutna klienter.

För Flash Player och Adobe AIR-innehåll utanför programsandlådan kräver XML-socketanslutningar en socketprincipfil på målservern. Mer information finns i ”Webbplatsinställningar (principfiler)” på sidan 1056 och ”Ansluta till socketar” på sidan 1073.

Klassen XMLSocket kan inte gå igenom brandväggar automatiskt, eftersom XMLSocket i motsats till RTMP (Real-Time Messaging Protocol) saknar HTTP-tunnelkapacitet. Om du måste använda HTTP-tunnel bör du i stället använda Flash Remoting eller Flash Media Server (som har stöd för RTMP).

Nedanstående begränsningar avser hur och var innehåll i Flash Player eller i ett AIR-program utanför programsäkerhetssandlådan kan ansluta till servern med hjälp av ett XMLSocket-objekt:

- När det gäller innehåll utanför säkerhetssandlådan för programmet kan metoden `XMLSocket.connect()` ansluta endast till TCP-portnummer som är större än eller lika med 1024. En konsekvens av denna begränsning är att serverns daemon-processer som kommunicerar med `XMLSocket`-objektet måste tilldelas till portnummer som är större än eller lika med 1024. Portnummer under 1024 används ofta av systemtjänster, till exempel FTP (21), Telnet (23), SMTP (25), HTTP (80) och POP3 (110). Därför kan de av säkerhetsskäl inte användas i XMLSocket-objekt. Portnummerbegränsningen förhindrar att dessa resurser används på felaktigt sätt eller missbrukas.
- När det gäller innehåll utanför säkerhetssandlådan för programmet kan metoden `XMLSocket.connect()` ansluta endast till datorer i den domän där innehållet finns. (Begränsningen är identisk med säkerhetsreglerna för `URLLoader.load()`.) För att kunna ansluta till en server-daemon som körs i en annan domän än den där innehållet finns måste du skapa en korsdomänprincipfil på servern som tillåter åtkomst från specifika domäner. Mer information om korsdomänprincipfiler finns i ”AIR-säkerhet” på sidan 1081.

Obs! *Det kan vara svårt att ställa in en server för att kommunicera med XMLSocket-objektet. Om programmet inte kräver realtidsinteraktivitet använder du klassen URLLoader i stället för XMLSocket.*

Du kan använda metoderna `XMLSocket.connect()` och `XMLSocket.send()` för klassen XMLSocket för att överföra XML till och från en server via en socketanslutning. Med metoden `XMLSocket.connect()` upprättas en socketanslutning till en webbserverport. Med metoden `XMLSocket.send()` skickas ett XML-objekt till den server som har angetts i socketanslutningen.

När du anropar metoden `XMLSocket.connect()` öppnar programmet en TCP/IP-anslutning till servern och anslutningen förblir öppen tills något av följande inträffar:

- Metoden `XMLSocket.close()` för klassen XMLSocket class anropas.
- Det finns inte några fler referenser till XMLSocket-objektet.
- Flash Player avslutas.
- Anslutningen avbryts (till exempel genom att modemet kopplas från).

Ansluta till en server med klassen XMLSocket

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att kunna skapa en socketanslutning måste du skapa ett program på servern som väntar på en socketanslutningsbegäran och skickar ett svar till Flash Player eller AIR-programmet. Den här typen av program på servern kan skrivas i AIR eller i ett annat programmeringsspråk, till exempel Java, Python eller Perl. För att klassen XMLSocket ska kunna användas måste servern ha en daemon-process som förstår det enkla protokollet som klassen XMLSocket använder:

- XML-meddelanden skickas över en TCP/IP-strömsocketanslutning med full duplex.
- Varje XML-meddelande är ett komplett XML-dokument som avslutas av en nollbyte (0).
- Ett obegränsat antal XML-meddelanden kan skickas och tas emot via en enda XMLSocket-anslutning.

Skapa och ansluta till en Java XML-socketserver

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Nedanstående kod visar en enkel XMLSocket-server skriven i Java som godtar inkommande anslutningar och visar de mottagna meddelandena i kommandotolksfönstret. En ny server skapas som standard på port 8080 på din lokala dator, men du kan ange ett annat portnummer när du startar servern från kommandoraden.

Skapa ett nytt textdokument och lägg till följande kod:

```
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)
        {
            // Catch exception and keep going.
        }

        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);
    }
}
```

Spara dokumentet på hårddisken som SimpleServer.java och kompilera det med hjälp av en Java-kompilator, vilket skapar en Java-klassfil med namnet SimpleServer.class.

Du kan starta XMLSocket-servern genom att öppna en kommandotolk och skriva `java SimpleServer`. Filen SimpleServer.class kan finnas på valfri plats på din dator eller i nätverket. Den behöver inte placeras i rotkatalogen på webbservern.

Om du inte kan starta servern för att filerna inte finns i Java-klassökvägen, kan du försöka att starta servern med `java -classpath . SimpleServer`.

För att ansluta till XMLSocket från ditt program måste du skapa en ny instans av klassen XMLSocket och anropa metoden `XMLSocket.connect()` när du skickar ett värdnamn och portnummer enligt följande:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Varje gång du tar emot data från servern skickas `data`-händelsen (`flash.events.DataEvent.DATA`).

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Om du vill skicka data till XMLSocket-servern använder du metoden `XMLSocket.send()` och skickar ett XML-objekt eller -sträng. Flash Player konverterar den angivna parametern till ett String-objekt och skickar innehållet till XMLSocket-servern följt av en nollbyte (0).

```
xmlsock.send(xmlFormattedData);
```

Metoden `XMLSocket.send()` returnerar inte ett värde som anger om data kunde skickas. Om ett fel uppstår när data skickas returneras ett IOError-fel.

Varje meddelande som du skickar till XML-socketservern måste sluta med ett radmatningstecken (`\n`).

Mer information finns i XMLSocket.

Serversocketar

Adobe AIR 2 och senare

Använd klassen ServerSocket för att tillåta att andra processer ansluter till ditt program med hjälp av en TCP-socket (Transport Control Protocol). Anslutningsprocessen kan köras på den lokala datorn eller på någon annan dator som är ansluten till nätverket. När ett ServerSocket-objekt tar emot en anslutningsförfrågan skickas en `connect`-händelse. Objektet ServerSocketConnectEvent som skickas med händelsen innehåller ett Socket-objekt. Du kan använda det här socketobjektet för kommande kommunikation med andra processer.

Så här lyssnar du efter inkommande socketanslutningar:

1. Skapa ett ServerSocket-objekt och bind det till en lokal port
2. Lägg till händelseavlyssnare för händelsen `connect`
3. Anropa metoden `listen()`
4. Svara på händelsen`connect`, som tillhandahåller ett Socket-objekt för alla inkommande anslutningar

ServerSocket-objektet fortsätter att lyssna efter nya anslutningar tills du anropar metoden `close()`.

Följande kodexempel illustrerar hur ett socketserverprogram skapas. Exemplet lyssnar efter inkommande anslutningar på port 8087. När en anslutning tas emot skickas ett meddelande (strängen "Connected.") av exemplet till klientens socket. Sedan skickas meddelanden som servern tagit emot tillbaka till klienten.

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.ServerSocketConnectEvent;
    import flash.net.ServerSocket;
    import flash.net.Socket;

    public class ServerSocketExample extends Sprite
    {
        private var serverSocket:ServerSocket;
        private var clientSockets:Array = new Array();

        public function ServerSocketExample()
        {
            try
            {
                // Create the server socket
                serverSocket = new ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );

                // Bind to local port 8087
                serverSocket.bind( 8087, "127.0.0.1" );

                // Listen for connections
                serverSocket.listen();
                trace( "Listening on " + serverSocket.localPort );
```

```
            }
            catch(e:SecurityError)
            {
                trace(e);
            }
        }

        public function connectHandler(event:ServerSocketConnectEvent):void
        {
            //The socket is provided by the event object
            var socket:Socket = event.socket as Socket;
            clientSockets.push( socket );

            socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
            socket.addEventListener( Event.CLOSE, onClientClose );
            socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );

            //Send a connect message
            socket.writeUTFBytes("Connected.");
            socket.flush();

            trace( "Sending connect message" );
        }

        public function socketDataHandler(event:ProgressEvent):void
        {
            var socket:Socket = event.target as Socket

            //Read the message from the socket
            var message:String = socket.readUTFBytes( socket.bytesAvailable );
            trace( "Received: " + message);

            // Echo the received message back to the sender
            message = "Echo -- " + message;
            socket.writeUTFBytes( message );
            socket.flush();
            trace( "Sending: " + message );
        }

        private function onClientClose( event:Event ):void
        {
            trace( "Connection to client closed." );
            //Should also remove from clientSockets array...
        }

        private function onIOError( errorEvent:IOErrorEvent ):void
        {
            trace( "IOError: " + errorEvent.text );
        }

        private function onClose( event:Event ):void
        {
            trace( "Server socket closed by OS." );
        }
}}
```

Mer information finns i:

- ServerSocket
- ServerSocketConnectEvent
- Socket

UDP-socketar (AIR)

Adobe AIR 2 och senare

Universal Datagram Protocol (UDP) är ett sätt att utbyta meddelanden över en tillståndslös nätverksanslutning. Med UDP lämnas ingen garanti för att meddelanden levereras i ordning eller att de levereras över huvud taget. Med UDP brukar operativsystemets nätverkskod använda mindre tid för att konvertera, spåra och bekräfta meddelanden. Därför anländer UDP-meddelanden vanligtvis till målprogrammet med kortare fördröjning än vad TCP-meddelanden gör.

UDP-socketkommunikation kan vara till hjälp när du måste skicka realtidsinformation, till exempel uppdaterade positioner i ett spel eller ljudpaket i ett ljudchattprogram. I sådana program accepteras en viss dataförlust och låg överföringsfördröjning är viktigare än garanterad leverans. För så gott som alla andra syften är TCP-socketar ett bättre val.

AIR-programmet kan skicka och ta emot UDP-meddelanden med klasserna DatagramSocket och DatagramSocketDataEvent. Så här skickar du eller tar emot ett UDP-meddelande:

1. Skapa ett DatagramSocket-objekt
2. Lägg till en händelseavlyssnare för händelsen `data`
3. Bind socketen till en lokal IP-adress och port med hjälp av metoden `bind()`
4. Skicka meddelanden genom att anropa metoden `send()`, och överföra måldatorns IP-adress och port
5. Ta emot meddelanden genom att svara på händelsen `data`. Det DatagramSocketDataEvent-objekt som skickades för den här händelsen innehåller ett ByteArray-objekt med alla meddelandedata.

I följande kodexempel illustreras hur ett UDP-meddelande skickas och tas emot i ett program. I exemplet skickas ett enkelt meddelande som innehåller strängen "Hello." till måldatorn. Innehållet spåras också i samtliga mottagna meddelanden.

```
package
{
import flash.display.Sprite;
import flash.events.DatagramSocketDataEvent;
import flash.events.Event;
import flash.net.DatagramSocket;
import flash.utils.ByteArray;

public class DatagramSocketExample extends Sprite
{
    private var datagramSocket:DatagramSocket;

    //The IP and port for this computer
    private var localIP:String = "192.168.0.1";
    private var localPort:int = 55555;

    //The IP and port for the target computer
    private var targetIP:String = "192.168.0.2";
    private var targetPort:int = 55555;

    public function DatagramSocketExample()
    {
        //Create the socket
        datagramSocket = new DatagramSocket();
        datagramSocket.addEventListener( DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();

        //Create a message in a ByteArray
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    private function dataReceived( event:DatagramSocketDataEvent ):void
    {
        //Read the data from the datagram
        trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
}}
```

Kom ihåg följande när du använder UDP-socketar:

- Ett enskilt datapaket kan inte vara större än den minsta maximala överföringsenheten (MTU) i nätverkets gränssnitt eller i några nätverksnoder mellan avsändaren och mottagaren. Alla data i ByteArray-objektet som överförs till metoden send() skickas i ett enskilt paket. (I TCP bryts stora meddelanden upp i separata paket.)
- Det förekommer ingen handskakning mellan avsändaren och mottagaren. Meddelanden tas bort utan fel om målet inte finns eller inte har någon aktiv avlyssnare vid den angivna porten.

- När du använder metoden `connect()` ignoreras meddelanden som skickas från annan källa. Med en UDP-anslutning tillhandahålls endast lämplig paketfiltrering. Det innebär inte att det nödvändigtvis finns en giltig avlyssningsprocess vid målets adress och port.
- Ett nätverk kan översvämmas av UDP-trafik. Nätverksadministratörer kan behöva implementera kontroller av tjänstekvalitet om nätverket överbelastas. (I TCP finns inbyggda trafikkontroller för att minska effekten om nätverket överbelastas.)

Mer information finns i:

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

IPv6-adresser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player 9.0.115.0 och senare stöder IPv6 (Internet Protocol version 6). IPv6 är en version av Internetprotokollet med stöd för adresser med 128-bitar (en förbättring från den tidigare versionen IPv4 som har stöd för 32-bitar). Du kanske måste aktivera IPv6 i dina nätverksgränssnitt. Mer information finns i hjälpen för operativsystemet som dina data finns på.

Om IPv6 stöds i operativsystemet kan du ange numeriska IPv6-adresser uttryckligen i URL:er inom hakparenteser ([]) på följande sätt:

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Flash Player returnerar litterala IPv6-värden, enligt följande regler:

- Flash Player returnerar det långa formatet av strängen för IPv6-adresser.
- IP-värdet har inga förkortningar med dubbla kolon.
- Hexadecimala siffror anges enbart med gemener.
- IPv6-adresser omsluts med hakparentes ([]).
- Varje adresskvartett returneras som 0 till 4 hexadecimala siffror, där inledande nollor utelämnas.
- En adresskvartett med enbart nollor returneras som en enda nolla (utan dubbelt kolon) utom enligt vad som anges i följande lista med undantag.

IPv6-värdena som Flash Player returnerar har följande undantag:

- En icke angiven IPv6-adress (enbart nollor) returneras som [::].
- Loopback- eller localhost-IPv6-adressen returneras som [::1].
- IPv4-mappade (konverterade till IPv6) adresser returneras som [::ffff:a.b.c.d], där a.b.c.d är ett vanligt IPv4-värde med decimalpunkt.
- IPv4-kompatibla adresser returneras som [::a.b.c.d], där a.b.c.d är ett vanligt IPv4-värde med decimalpunkt.

Kapitel 44: HTTP-kommunikation

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Adobe® AIR®- och Adobe® Flash® Player-program kan kommunicera med HTTP-baserade servrar för inläsning av data, bilder och video och för överföring av meddelanden.

Fler hjälpavsnitt

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

Läsa in externa data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

ActionScript 3.0 innehåller funktioner för att läsa in data från externa källor. Dessa källor kan tillhandahålla statiskt innehåll såsom textfiler, eller dynamiskt innehåll som genereras av ett webbskript. Data kan vara formaterade på olika sätt, och ActionScript erbjuder funktionalitet för att avkoda och få tillgång till data. Du kan även skicka data till den externa servern som en del av hämtningen av data.

Använda klassen URLRequest

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Många API:er som läser in externa data använder klassen URLRequest för att definiera egenskaperna för den nödvändiga nätverksbegäran.

URLRequest-egenskaper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande egenskaper kan anges för ett URLRequest-objekt i alla säkerhetssandlådor:

Egenskap	Beskrivning
contentType	MIME-innehållstypen för alla data som skickas med URL-begäran. Om inget contentType har angetts skickas värden som `application/x-www-form-urlencoded`.
data	Ett objekt som innehåller data som ska överföras med URL-begäran.
digest	En sträng som ger en unik identifiering av den signerade Adobe-plattformskomponent som ska lagras i (eller hämtas från) Adobe® Flash® Player-cachen.
method	Metoden för en HTTP-begäran, till exempel GET eller POST. (Innehåll som körs i säkerhetsdomänen för AIR-programmet kan ange andra strängar än `GET` eller `POST` som `method`-egenskap. Alla HTTP-verb är tillåtna och standardmetoden är `GET`. Se även "AIR-säkerhet" på sidan 1081.)
requestHeaders	Arrayen med rubriker för HTTP-begäranden som ska läggas till i HTTP-begäran. Observera att tillstånd för att ange vissa huvuden är begränsat i Flash Player och i AIR-innehåll som körs utanför programmets säkerhetssandlåda.
url	Anger den URL som ska begäras.

I AIR kan du ange ytterligare egenskaper för klassen URLRequest som endast är tillgängliga för AIR-innehåll som körs i programmets säkerhetssandlåda. Innehåll i programsandlådan kan också definiera URL:er som använder nya URL-scheman (utöver standardscheman som `file` och `http`).

Egenskap	Beskrivning
followRedirects	Anger om omdirigeringar ska följas (`true`, standardvärdet) eller inte (`false`). Detta stöds endast i sandlådan för AIR-programmet.
manageCookies	Anger om HTTP-protokollstacken ska hantera cookies (`true`, standardvärdet) eller inte (`false`) för denna begäran. Den här egenskapen kan endast anges i sandlådan för AIR-programmet.
authenticate	Anger om autentiseringsbegäranden ska hanteras (`true`) för denna begäran. Den här egenskapen kan endast anges i sandlådan för AIR-programmet. Standardinställningen är att begäranden ska autentiseras, vilket kan innebära att en dialogruta för autentisering visas om servern kräver autentiseringsuppgifter. Du kan också ange användarnamn och lösenord med klassen URLRequestDefaults, se "Ange standardvärden för URLRequest (endast AIR)" på sidan 814.
cacheResponse	Anger om svarsdata ska cachelagras för den här begäran. Den här egenskapen kan endast anges i sandlådan för AIR-programmet. Standardvärdet är att svaret cachelagras (`true`).
useCache	Anger om den lokala cachen ska genomsökas innan data hämtas med det här URLRequest-objektet. Den här egenskapen kan endast anges i sandlådan för AIR-programmet. Standardvärdet (`true`) är att den lokala cacheversionen används, om en sådan är tillgänglig.
userAgent	Anger den användaragentsträng som ska användas i HTTP-begäran.

Obs! *Klassen HTMLLoader har relaterade egenskaper för inställningar som rör innehåll som läses in av ett HTMLLoader-objekt. Mer information finns i "Om klassen HTMLLoader" på sidan 985.*

Ange standardvärden för URLRequest (endast AIR)

Adobe AIR 1.0 och senare

Med klassen URLRequestDefaults kan du definiera programspecifika standardinställningar för URLRequest-objekt. I följande kod anges till exempel standardvärdena för egenskaperna `manageCookies` och `useCache`. För alla nya URLRequest-objekt används dessa angivna värden för egenskaperna istället för standardvärden:

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
```

Obs! *Klassen URLRequestDefaults anges endast för innehåll som körs i Adobe AIR. Det stöds inte i Flash Player.*

Klassen URLRequestDefaults innehåller en `setLoginCredentialsForHost()`-metod som gör att du kan ange ett användarnamn och lösenord som används som standard för en specifik värd. Värden definieras i parametern hostname för metoden och kan vara en domän, till exempel `"www.example.com"`, eller en domän och ett portnummer, till exempel `"www.example.com:80"`. Observera att `"example.com"`, `"www.example.com"` och `"sales.example.com"` alla behandlas som unika värdar.

Dessa användaruppgifter används endast om servern kräver dem. Om användaren redan har autentiserats (till exempel i dialogrutan för autentisering) ändras inte den autentiserade användaren vid anrop av metoden `setLoginCredentialsForHost()`.

I koden nedan anges standardvärden för användarnamn och lösenord som används när en begäran skickas till www.example.com:

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

Inställningarna för URLRequestDefaults gäller endast för den aktuella programdomänen, med ett undantag. Inloggningsuppgifterna som skickas till metoden `setLoginCredentialsForHost()` används för begäran i alla programdomäner inom AIR-programmet.

Du hittar mer information i avsnittet om klassen URLRequestDefaults i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

URI-scheman

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Standardscheman för URI, som beskrivs här, kan användas i en begäran från alla typer av säkerhetssandlådor:

http: och https:
Använd dessa för vanliga Internetadresser (på samma sätt som de används i en webbläsare).

file:
Använd `file:` om du vill ange en adress till en fil som finns på det lokala filsystemet. Till exempel:

```
file:///c:/AIR Test/test.txt
```

I AIR kan du också använda följande scheman när du definierar en URL för innehåll som körs i säkerhetssandlådan för programmet:

app:
Använd `app:` om du vill ange en relativ sökväg till det installerade programmets rotkatalog. Följande sökväg pekar till exempel på en resursunderkatalog i katalogen för det installerade programmet:

```
app:/resources
```

När ett AIR-program startas med AIR Debug Launcher (ADL) är programkatalogen den katalog som innehåller programbeskrivningsfilen.

I URL (och `url`-egenskapen) för ett File-objekt som har skapats med `File.applicationDirectory` används URI-schemat `app`, enligt följande exempel:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("assets");
trace(dir.url); // app:/assets
```

app-storage:

Använd `app-storage:` om du vill ange en relativ sökväg till programmets datalagringskatalog. För varje installerat program (och användare) skapar AIR en unik programlagringskatalog, där du kan lagra data som gäller för det programmet. Följande sökväg pekar till exempel på en prefs.xml-fil i en inställningsunderkatalog till programlagringskatalogen:

```
app-storage:/settings/prefs.xml
```

I URL (och `url`-egenskapen) för ett File-objekt som har skapats med `File.applicationStorageDirectory` används URI-schemat `app-storage`, enligt följande exempel:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
trace(dir.prefsFile); // app-storage:/prefs.xml
```

mailto:

Du kan använda mailto-schemat i URLRequest-objekt som skickas till `navigateToURL()`-funktionen. Läs mer i ”Öppna en URL i ett annat program” på sidan 830.

Du kan använda ett URLRequest-objekt som har något av dessa URI-scheman när du definierar URL-begäran för ett antal olika objekt, till exempel ett FileStream- eller Sound-objekt. Du kan också använda dessa scheman i HTML-innehåll som körs i AIR. Du kan till exempel använda dem i `img`-taggens `src`-attribut.

Du kan dock endast använda dessa AIR-specifika URI-scheman (`app:` och `app-storage:`) i innehåll i säkerhetssandlådan för programmet. Mer information finns under ”AIR-säkerhet” på sidan 1081.

Ange URL-variabler

När du lägger till variabler i en URL-sträng direkt kan det vara lättare att använda klassen URLVariables för att definiera de variabler som krävs för en begäran.

Du kan lägga till parametrar i ett URLVariables-objekt på tre sätt:

- I URLVariables-konstruktorn
- Med metoden `URLVariables.decode()`
- Som dynamiska egenskaper för själva URLVariables-objektet

I följande exempel visas alla tre metoder och också hur du tilldelar variablerna till ett URLRequest-objekt:

```
var urlVar:URLVariables = new URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest:URLRequest = new URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

När du definierar variabler inom URLVariables-konstruktorn eller i metoden `URLVariables.decode()`, ska du se till att du URL-kodar alla tecken som har en särskild betydelse i en URI-sträng. Om du till exempel använder ett &-tecken i ett parameternamn eller värde måste du koda &-tecknet genom att ändra det från `&` till `%26` eftersom &-tecknet fungerar som en avgränsare för parametrar. Du kan använda högnivåfunktionen `encodeURIComponent()` för detta.

Använda klassen URLLoader

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen URLLoader kan du skicka en begäran till en server och få åtkomst till den information som skickas tillbaka. Du kan också använda klassen URLLoader för att få åtkomst till filer på det lokala filsystemet om lokal filåtkomst tillåts (till exempel i sandlådan lokal-med-filsystem i Flash Player och i AIR-programsandlådan). Med klassen URLLoader laddas data ned från en URL som text, binära data eller URL-kodade variabler. Klassen URLLoader skickar händelser som `complete`, `httpStatus`, `ioError`, `open`, `progress` och `securityError`.

Händelsehanteringsmodellen i ActionScript 3.0 skiljer sig väsentligen från ActionScript 2.0-modellen, som använde händelsehanterarna `LoadVars.onData`, `LoadVars.onHTTPStatus` och `LoadVars.onLoad`. Mer information om hur du hanterar händelser i ActionScript 3.0 finns i ”Hantera händelser” på sidan 125

Hämtade data är inte tillgängliga förrän hämtningen har slutförts. Du kan följa hämtningsförloppet (inlästa byte av totalt antal byte) genom att avlyssna när händelsen `progress` skickas. Men om det går tillräckligt fort att läsa in en fil kanske inte någon `progress`-händelse skickas. När en fil har laddats ned skickas händelsen `complete`. Genom att ange URLLoader-egenskapen `dataFormat` kan du välja att ta emot data som text, binära rådata eller som ett URLVariables-objekt.

Metoden `URLLoader.load()` (och eventuellt konstruktorn för klassen URLLoader) har en enda parameter, `request`, som är ett URLRequest-objekt. Ett URLRequest-objekt innehåller all information för en enskild HTTP-begäran, till exempel mål-URL, begärandemetod (`GET` eller `POST`), annan huvudinformation samt MIME-typ.

Om du till exempel vill överföra ett XML-paket till ett skript på en server kan du använda följande kod:

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <clock>
        <time>{secondsUTC}</time>
    </clock>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Ovanstående kodutdrag skapar ett XML-dokument med namnet `dataXML` som innehåller det XML-paket som ska skickas till servern. I exemplet anges att URLRequest-egenskapen `contentType` ska vara `"text/xml"` och XML-dokumentet tilldelas URLRequest-egenskapen `data`. Slutligen skapar exemplet ett URLLoader-objekt och skickar begäran till fjärrskriptet med metoden `load()`.

Använda klassen URLStream

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen URLStream ger åtkomst till hämtade data samtidigt som de hämtas. Med klassen URLStream kan du även stänga en ström innan hämtningen har slutförts. De data som hämtats är tillgängliga som binära rådata.

Vid läsning av data från ett URLStream-objekt använder du egenskapen `bytesAvailable` om du vill avgöra om tillräcklig mängd data finns tillgänglig för inläsningen. Ett EOFError-undantag genereras om du försöker att läsa mer data än vad som finns tillgängligt.

Händelsen httpResponseStatus (AIR)

I Adobe AIR skickar klassen URLStream en `httpResponseStatus`-händelse tillsammans med händelsen `httpStatus`. Händelsen `httpResponseStatus` levereras före alla eventuella svarsdata. Händelsen `httpResponseStatus` (som representeras av klassen HTTPStatusEvent) innehåller en `responseURL`-egenskap, som är den URL som svaret returnerades från, och en `responseHeaders`-egenskap, som är en array med URLRequestHeader-objekt som representerar de svarshuvuden som svaret returnerade.

Läsa in data från externa dokument

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar dynamiska program kan det vara användbart att läsa in data från externa filer eller från skript på servern. På det sättet kan du skapa dynamiska program utan att behöva redigera eller kompilera om programmet. Om du till exempel skapar ett program för ”dagens tips” kan du skriva ett skript på servern som hämtar ett slumpmässigt tips från en databas och sparar det i en textfil en gång om dagen. Då kan programmet läsa in innehållet från en statisk textfil i stället för att varje gång skicka en begäran till databasen.

Följande kodfragment skapar ett URLRequest- och ett URLLoader-objekt, som läser in innehållet i en extern textfil med namnet params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Standardinställningen innebär, att om du inte definierar någon förfrågningsmetod, läser Flash Player och Adobe AIR in innehållet med hjälp av HTTP `GET`-metoden. Om du vill skicka begäran med hjälp av `POST`-metoden anger du `POST` för egenskapen `request.method` med hjälp av den statiska konstanten `URLRequestMethod.POST` enligt koden nedan:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

Det externa dokumentet params.txt som läses in under körningen innehåller följande data:

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

Filen innehåller två parametrar: `monthNames` och `dayNames`. Varje parameter innehåller en kommaavgränsad lista som analyseras som strängar. Du kan dela in listan i en array med hjälp av metoden `String.split()`.

Undvik reserverade ord eller språkkonstruktioner som variabelnamn i externa datafiler, eftersom det försvårar läsningen och felsökningen av din kod.

När data har lästs in skickas händelsen `complete` och innehållet i det externa dokumentet blir tillgängligt för användning i `data`-egenskapen för URLLoader som i följande kod:

```
function completeHandler(event)
{
    var loader2 = event.target;
    air.trace(loader2.data);
}
```

Om fjärrdokumentet innehåller namnvärdespar kan du analysera data med hjälp av klassen URLVariables genom att ange innehållet i den inlästa filen enligt följande:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Varje namnvärdespar från den externa filen skapas som en egenskap i URLVariables-objektet. Varje egenskap i variabelobjektet i ovanstående kodexempel behandlas som en sträng. Om värdet i namnvärdesparet är en lista över objekt, kan du konvertera strängen till en array genom att anropa metoden `String.split()` enligt följande:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

Om du läser in numeriska data från externa datafiler konverterar du värden till numeriska värden med hjälp av en högnivåfunktion, till exempel `int()`, `uint()` eller `Number()`.

I stället för att läsa in innehållet i fjärrfilen som en sträng och skapa ett nytt URLVariables-objekt, kan du ställa in egenskapen `URLLoader.dataFormat` på en av de statiska egenskaperna som finns i klassen URLLoaderDataFormat. Det finns tre möjliga värden för egenskapen `URLLoader.dataFormat`:

- `URLLoaderDataFormat.BINARY`: Egenskapen `URLLoader.data` kommer att innehålla binära data lagrade i ett ByteArray-objekt.
- `URLLoaderDataFormat.TEXT`: Egenskapen `URLLoader.data` kommer att innehålla text i ett String-objekt.
- `URLLoaderDataFormat.VARIABLES`: Egenskapen `URLLoader.data` kommer att innehålla URL-kodade variabler lagrade i ett URLVariables-objekt.

Följande kod visar hur inställning av egenskapen `URLLoader.dataFormat` på `URLLoaderDataFormat.VARIABLES` gör att du automatiskt kan analysera inlästa data till ett URLVariables-objekt:

```
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);
        }
    }
}
```

Obs! *Standardvärdet för* `URLLoader.dataFormat` *är* `URLLoaderDataFormat.TEXT`.

Som nedanstående exempel visar är inläsning av XML från en extern fil detsamma som att läsa in URLVariables. Du kan skapa en URLRequest-instans och en URLLoader-instans och sedan använda dem för att ladda ned ett XML-fjärrdokument. När filen har laddats ned skickas händelsen `Event.COMPLETE` och innehållet i den externa filen konverteras till en XML-instans, som du kan analysera med hjälp av XML-metoder och -egenskaper.

```
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());
        }
    }
}
```

Kommunicera med externa skript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen URLVariables kan inte bara användas för att läsa in externa filer. Du kan även använda den för att skicka variabler till ett skript på servern och för att bearbeta serverns svar. Det här är användbart om du till exempel programmerar ett spel och vill skicka användarens resultat till en server för att beräkna om resultatet ska läggas in i listan med bästa resultat, eller om en användares inloggningsinformation ska skickas till en server för validering. Ett skript på en server kan bearbeta användarnamnet och lösenordet, validera informationen mot en databas och sedan returnera en bekräftelse som anger om informationen är giltig.

Nedanstående utdrag skapar ett URLVariables-objekt med namnet `variables`, som skapar en ny variabel med namnet `name`. Sedan skapas ett URLRequest-objekt som anger webbadressen för skriptet på servern som variablerna ska skickas till. Sedan anger du egenskapen `method` för URLRequest-objektet så att variablerna skickas som en HTTP `POST`-begäran. Om du vill lägga till URLVariables-objektet till URL-begäran anger du det tidigare skapade URLVariables-objektet för URLRequest-objektets `data`-egenskap. Slutligen skapas URLLoader-instansen och metoden `URLLoader.load()` anropas, vilket initierar begäran.

```
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);
}
```

Följande kod rymmer innehållet i Adobe ColdFusion®-dokumentet greeting.cfm från föregående exempel:

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Webbtjänstbegäran

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det finns en mängd olika HTTP-baserade webbtjänster. De vanligaste är:

- REST
- XML-RPC
- SOAP

Om du vill använda en webbtjänst i ActionScript 3 skapar du ett URLRequest-objekt, konstruerar anropet till webbtjänsten antingen med hjälp av URL-variabler eller med ett XML-dokument, och skickar sedan anropet till tjänsten med ett URLLoader-objekt. Flex-ramverket innehåller flera klasser som gör det lättare att använda webbtjänster och som är särskilt användbara för åtkomst av avancerade SOAP-tjänster. Från och med Flash Professional CS3 kan du använda Flex-klasserna både i program som utvecklats med Flash Professional och i program som utvecklats i Flash Builder.

I HTML-baserade AIR-program kan du antingen använda klasserna URLRequest och URLLoader eller Javascript-klassen XMLHttpRequest. Om du vill kan du också skapa ett SWF-bibliotek som visar Flex-ramverkets webbtjänstkomponenter för din Javascript-kod.

När ett program körs i en webbläsare kan du endast använda webbtjänster på samma Internetdomän som den SWF som anropas, såvida inte servern som är värd för webbtjänsten också har en korsdomänprincipfil som tillåter åtkomst från andra domäner. En metod som ofta används när en korsdomänprincipfil inte finns tillgänglig är att använda en proxy för begäranden via en egen server. Adobe Blaze DS och Adobe LiveCycle stöder en proxy för webbtjänster.

I AIR-program krävs inte någon korsdomänprincipfil om webbtjänstanropet kommer från programmets säkerhetssandlåda. AIR-programinnehåll kommer aldrig från en fjärrdomän och kan därför inte ingå i den typen av attacker som korsdomänprinciper förhindrar. I HTML-baserade AIR-program kan innehåll i programmets säkerhetssandlåda göra XMLHttpRequests över olika domäner. Du kan tillåta att innehåll från andra säkerhetssandlådor kan göra XMLHttpRequests över olika domäner så länge som innehållet läses in i en så kallad iframe.

Fler hjälpavsnitt

"Webbplatsinställningar (principfiler)" på sidan 1056

Adobe BlazeDS

Adobe LiveCycle ES2

REST-arkitektur

XML-RPC

SOAP-protokoll

REST-webbtjänstbegäran

Flash Player 9 och senare, Adobe AIR 1.0 och senare

REST-webbtjänster använder HTTP-metoder för att ange grundläggande åtgärder och URL-variabler för information om åtgärderna. En begäran för att hämta data för ett objekt kan till exempel använda GET-verbet och URL-variabler för att ange ett metodnamn och ett objekt-ID. URL-strängen kan se ut så här:

```
http://service.example.com/?method=getItem&id=d3452
```

För att få åtkomst till en REST-webbtjänst med ActionScript kan du använda klasserna URLRequest, URLVariables och URLLoader. I Javascript-koden i ett AIR-program kan du även använda en XMLHttpRequest-begäran.

Ett vanligt upplägg för programmeringen av ett REST-webbtjänstanrop i ActionScript kan se ut enligt följande:

1. Skapa ett URLRequest-objekt.
2. Ange tjänstens URL och HTTP-metoden för det begärda objektet.
3. Skapa ett URLVariables-objekt.
4. Ange parametrarna för anropet av tjänsten som dynamiska egenskaper för variabelobjektet.
5. Tilldela variabelobjektet till dataegenskapen för det begärda objektet.
6. Skicka anropet till tjänsten med ett URLLoader-objekt.
7. Hantera händelsen `complete` som skickas av URLLoader och anger att anropet av tjänsten har slutförts. Det kan också vara bra att lyssna efter olika felhändelser som kan skickas av URLLoader-objektet.

Ta som exempel en webbtjänst som exponerar en testmetod som skickar anropsparametrarna tillbaka till begäran som ett eko. Följande ActionScript-kod kan användas för att anropa tjänsten:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;

private var requestor:URLLoader = new URLLoader();
public function restServiceCall():void
{
    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/" );
    request.method = URLRequestMethod.GET;

    //Add the URL variables
    var variables:URLVariables = new URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    requestor = new URLLoader();
    requestor.addEventListener( Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
private function httpRequestComplete( event:Event ):void
{
    trace( event.target.data );
}

private function httpRequestError( error:ErrorEvent ):void{
    trace( "An error occured: " + error.message );
}
```

I Javascript inom ett AIR-program kan du göra samma begäran med hjälp av XMLHttpRequest-objektet:

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

XML-RPC-webbtjänstbegäran

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En XML-RPC-webbtjänst tar sina anropsparametrar som ett XML-dokument istället för som en uppsättning URL-variabler. För att utföra en transaktion med en XML-RPC-webbtjänst skapar du ett korrekt formaterat XML-meddelande och skickar det till webbtjänsten med HTTP-metoden `POST`. Därutöver bör du ange `Content-Type` som begärandehuvud så att servern hanterar data för begäran som XML-data.

I följande exempel visas hur du använder samma webbtjänstanrop som visades i REST-exemplet, men den här gången som en XML-RPC-tjänst:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
public function xmlRPCRequest():void
{
    //Create the XML-RPC document
    var xmlRPC:XML = <methodCall>
                                    <methodName></methodName>
                                    <params>
                                        <param>
                                            <value>
                                                <struct/>
                                            </value>
                                        </param>
                                    </params>
                                </methodCall>;

    xmlRPC.methodName = "test.echo";

    //Add the method parameters
    var parameters:Object = new Object();
    parameters.api_key = "123456ABC";
    parameters.message = "Able was I, ere I saw Elba.";

    for( var propertyName:String in parameters )
    {
        xmlRPC..struct.member[xmlRPC..struct.member.length + 1] =
            <member>
                <name>{propertyName}</name>
                <value>
                    <string>{parameters[propertyName]}</string>
                </value>
            </member>;
    }

    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/xml-rpc/" );
    request.method = URLRequestMethod.POST;
    request.cacheResponse = false;
    request.requestHeaders.push(new URLRequestHeader("Content-Type", "application/xml"));
```

```
    request.data = xmlRPC;

    //Initiate the request
    requestor = new URLLoader();
    requestor.dataFormat = URLLoaderDataFormat.TEXT;
    requestor.addEventListener( Event.COMPLETE, xmlRPCRequestComplete );
    requestor.addEventListener( IOErrorEvent.IO_ERROR, xmlRPCRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, xmlRPCRequestError );
    requestor.load( request );
}

private function xmlRPCRequestComplete( event:Event ):void
{
    trace( XML(event.target.data).toXMLString() );
}

private function xmlRPCRequestError( error:ErrorEvent ):void
{
    trace( "An error occurred: " + error );
}
```

WebKit i AIR stöder inte E4X-syntax, och metoden som användes för att skapa XML-dokumentet i föregående exempel fungerar inte i Javascript-kod. Istället måste du skapa XML-dokumentet med hjälp av DOM-metoder eller skapa dokumentet som en sträng och använda klassen Javascript DOMParser för att konvertera strängen till XML.

I följande exempel används DOM-metoder för att skapa ett XML-RPC-meddelande och en XMLHttpRequest för att utföra webbtjänsttransaktionen:

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );

    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
```

```
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );

            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
    }
    if( rootNode.childElementCount > 0 )
```

```
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

SOAP-webbtjänstbegäran

Flash Player 9 och senare, Adobe AIR 1.0 och senare

SOAP bygger på det allmänna XML-RPC-webbtjänstkonceptet och ger rikare, dock mer komplexa möjligheter för överföring av typdata. SOAP-webbtjänster innehåller vanligtvis en WSDL-fil (Web Service Description Language) som anger webbtjänstens anrop, datatyper och tjänst-URL. ActionScript 3 har inte något direkt stöd för SOAP, men du kan skapa ett SOAP XML-meddelande på egen hand, skicka den till servern och sedan tolka resultaten. Men förutom för de allra enklaste SOAP-webbtjänsterna kan du förmodligen spara en hel del utvecklingstid genom att använda ett befintligt SOAP-bibliotek.

Flex-ramverket innehåller bibliotek för åtkomst av SOAP-webbtjänster. I Flash Builder inkluderas biblioteket, rpc.swc, automatiskt i Flex-projekt eftersom det är en del av Flex-ramverket. I Flash Professional kan du lägga till Flex framework.swc och rpc.swc i biblioteksökvägen för ett projekt och sedan få åtkomst till Flex-klasserna med ActionScript.

Fler hjälpavsnitt

Using the Flex web service component in Flash Professional (Använda Flex-webbtjänstkomponenten i Flash Professional)

Cristophe Coenraets: Real-time Trader Desktop for Android (Realtidsprogram på Android för börsmäklare)

Öppna en URL i ett annat program

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan öppna en URL i en webbläsare eller ett annat program med hjälp av funktionen `navigateToURL()`. För innehåll som körs i AIR öppnar funktionen `navigateToURL()` sidan i systemets standardwebbläsare.

För det URLRequest-objekt du skickar som `request`-parameter för den här funktionen används endast `url`-egenskapen.

Den första parametern i funktionen `navigateToURL()` (parametern `navigate`) är ett URLRequest-objekt (se ”Använda klassen URLRequest” på sidan 813). Den andra parametern är en valfri `window`-parameter, i vilken du kan ange fönsternamnet. Följande kod öppnar till exempel webbsidan www.adobe.com:

```
var url:String = "http://www.adobe.com";
var urlReq:URLRequest = new URLRequest(url);
navigateToURL(urlReq);
```

Obs! *När funktionen `navigateToURL()` används behandlas ett URLRequest-objekt som använder POST-metoden (ett objekt med en `method`-egenskap som har värdet `URLRequestMethod.POST`) på samma sätt som GET-metoden.*

När du använder funktionen `navigateToURL()` tillåts URI-scheman baserat på säkerhetssandlådan för koden som anropar funktionen `navigateToURL()`.

Vissa API:er tillåter att du startar innehåll i en webbläsare. Av säkerhetsskäl är vissa URI-scheman inte tillåtna när du använder dessa API:er i AIR. Listan över förbjudna scheman beror på säkerhetssandlådan för den kod som använder API:n. (Mer information om säkerhetssandlådor finns i ”AIR-säkerhet” på sidan 1081.)

Programsandlåda (endast AIR)

Valfritt URI-schema kan användas i URL som startas av innehåll som körs i AIR-programmets sandlåda. Ett program måste vara registrerat för att URI-schemat ska kunna hanteras, eftersom inget annars händer med denna begäran. Följande scheman har stöd på många datorer och enheter:

- `http:`
- `https:`
- `file:`
- `mailto:` – AIR dirigerar dessa begäranden till systemets registrerade e-postprogram.
- `sms:` – AIR dirigerar `sms:`-begäranden till standardprogrammet för textmeddelanden. URL-formatet måste följa de systemregler som gäller för programmet. På Android måste URI-scheman till exempel skrivas med gemener.

  ```
  navigateToURL( new URLRequest( "sms:+15555550101") );
  ```

- `tel:` – AIR dirigerar `tel:`-begäranden till standardprogrammet för telefoni. URL-formatet måste följa de systemregler som gäller för programmet. På Android måste URI-scheman till exempel skrivas med gemener.

  ```
  navigateToURL( new URLRequest( "tel:5555555555") );
  ```

- `market:` – AIR dirigerar `market:`-begäranden till Market-programmet som vanligtvis stöds på Android-enheter.

  ```
  navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
  navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
  ```

Där det tillåts i operativsystemet kan program definiera och registrera egna URI-scheman. Du kan skapa ett URI-schema för att starta programmet från AIR.

Fjärrsandlådor

Följande scheman är tillåtna. Använd dessa scheman på samma sätt som i en webbläsare.

- `http:`
- `https:`
- `mailto:` – AIR dirigerar dessa begäranden till systemets registrerade e-postprogram.

Alla andra URI-scheman tillåts inte.

Sandlådan lokal-med-fil

Följande scheman är tillåtna. Använd dessa scheman på samma sätt som i en webbläsare.

- `file:`
- `mailto:` – AIR dirigerar dessa begäranden till systemets registrerade e-postprogram.

Alla andra URI-scheman tillåts inte.

Sandlådan lokal-med-nätverk

Följande scheman är tillåtna. Använd dessa scheman på samma sätt som i en webbläsare.

- `http:`
- `https:`
- `mailto:` – AIR dirigerar dessa begäranden till systemets registrerade e-postprogram.

Alla andra URI-scheman tillåts inte.

Lokal-tillförlitlig sandlåda

Följande scheman är tillåtna. Använd dessa scheman på samma sätt som i en webbläsare.

- `file:`
- `http:`
- https:
- `mailto:` – AIR dirigerar dessa begäranden till systemets registrerade e-postprogram.

Alla andra URI-scheman tillåts inte.

Kapitel 45: Kommunicera med andra Flash Player- och AIR-instanser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen LocalConnection används för kommunikation mellan Adobe® AIR®-program och mellan SWF-innehåll som körs i webbläsaren. Du kan också använda klassen LocalConnection för kommunikation mellan ett AIR-program och SWF-innehåll som körs i webbläsaren. Med hjälp av klassen LocalConnection kan du skapa mångsidiga program som kan dela data mellan olika instanser av Flash Player och AIR.

Om klassen LocalConnection

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen LocalConnection kan du utveckla SWF-filer som skickar instruktioner till andra SWF-filer utan att använda metoden fscommand() eller JavaScript. LocalConnection-objekt kan bara kommunicera med andra SWF-filer som körs på samma klientdator, men de kan köras i olika program. En SWF-fil som körs i en webbläsare och en SWF-fil som körs i en projektor kan till exempel dela information. Projektorn har då den lokala informationen och den webbläsarbaserade SWF-filen ansluter via en fjärranslutning. (En projektor är en SWF-fil som har sparats i ett format som kan köras som ett fristående program, d.v.s.: Flash Player behöver inte vara installerat eftersom projektorn är inbäddad i den körbara filen.)

LocalConnection-objekt kan användas för kommunikation mellan SWF-filer med hjälp av olika ActionScript-versioner:

- LocalConnection-objekt som har skapats i ActionScript 3.0 kan kommunicera med LocalConnection-objekt som har skapats i ActionScript 1.0 eller 2.0.
- LocalConnection-objekt som har skapats i ActionScript 1.0 eller 2.0 kan kommunicera med LocalConnection-objekt som har skapats i ActionScript 3.0.

I Flash Player sker kommunikationen mellan LocalConnection-objekt som har olika versioner automatiskt.

Det enklaste sättet att använda ett LocalConnection-objekt på är att bara tillåta kommunikation mellan LocalConnection-objekt som finns i samma domän eller i samma AIR-program. På det sättet behöver du inte oroa dig för eventuella säkerhetsproblem. Om du måste tillåta kommunikation mellan olika domäner kan du dock vidta flera olika säkerhetsåtgärder. Mer information finns i avsnittet om parametern `connectionName` för metoden `send()` och i avsnitten om `allowDomain()` och `domain` för klassen LocalConnection i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Det går att använda LocalConnection-objekt för att skicka och ta emot data i en enda SWF-fil, men det är inget vi rekommenderar. Använd i stället delade objekt.

Du kan lägga till återkallningsmetoder till LocalConnection-objekt på något av följande tre sätt:

- Använda underklasser till klassen LocalConnection och lägga till metoder
- Ställa in egenskapen `LocalConnection.client` på ett objekt som implementerar metoderna
- Skapa en dynamisk klass som utökar LocalConnection och bifogar metoder dynamiskt

Det första sättet att lägga till återkallningsmetoder på är att utöka klassen LocalConnection. Du definierar metoderna i den anpassade klassen i stället för att lägga till dem dynamiskt till LocalConnection-instansen. Följande kod visar detta:

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

För att kunna skapa en ny instans av klassen CustomLocalConnection kan du använda följande kod:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

Det andra sättet att lägga till återkallningsmetoder på är att använda egenskapen `LocalConnection.client`. Då skapar du en anpassad klass och tilldelar en ny instans till egenskapen `client` som i följande kod:

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

Egenskapen `LocalConnection.client` anger vilka objektåterkallningsmetoder som ska anropas. I ovanstående kod ställdes egenskapen `client` in på en ny instans av en anpassad klass, CustomClient. Standardvärdet för egenskapen `client` är den aktuella LocalConnection-instansen. Du kan använda egenskapen `client` om du har två datahanterare som har samma uppsättning metoder, men fungerar på olika sätt. Det kan till exempel gälla ett program där en knapp i ett fönster växlar visningen i ett andra fönster.

Om du vill skapa klassen CustomClient använder du följande kod:

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Det tredje sättet att lägga till återkallningsmetoder på innebär att skapa en dynamisk klass och att sedan bifoga metoderna dynamiskt. Det påminner om sättet som klassen LocalConnection användes på i tidigare versioner av ActionScript, vilket följande kod visar:

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Återkallningsmetoder kan läggas till dynamiskt till den här klassen med hjälp av följande kod:

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

Det föregående sättet att lägga till återkallningsmetoder på rekommenderas inte, eftersom koden inte är särskild portabel. Dessutom skulle det här sättet att skapa lokala anslutningar på skapa prestandaproblem, eftersom det tar avsevärt mycket längre tid att få tillgång till dynamiska egenskaper jämfört med fasta (icke-dynamiska) egenskaper.

Egenskapen isPerUser

Egenskapen `isPerUser` lades till i Flash Player (10.0.32) och AIR (1.5.2) för att lösa en konflikt som inträffar när fler än en användare är inloggad på en Mac-dator. I andra operativsystem ignoreras egenskapen eftersom den lokala anslutningen alltid har omfattat endast enskilda användare. Egenskapen `isPerUser` ska vara inställd på `true` i ny kod. Det aktuella standardvärdet är emellertid `false` för bakåtkompabilitet. Standardvärdet kan ändras i framtida versioner av körningstider.

Skicka meddelanden mellan två program

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder klassen LocalConnection för kommunikation mellan olika AIR-program och mellan olika Adobe® Flash® Player-program (SWF) som körs i en webbläsare. Du kan också använda klassen LocalConnection för kommunikation mellan ett AIR-program och ett SWF-program som körs i en webbläsare.

Du kan till exempel ha flera Flash Player-instanser på en webbsida, eller låta en Flash Player-instans hämta data från en Flash Player-instans i ett popup-fönster.

Följande kod definierar ett LocalConnection-objekt som fungerar som en server och som accepterar inkommande LocalConnection-anrop från andra program:

```
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");
            }
        }
    }
}
```

Koden skapar först ett LocalConnection-objekt som heter `lc` och ställer in egenskapen `client` till ett objekt, `clientObject`. När ett annat program anropar en metod i den här LocalConnection-instansen, söker körningen efter metoden i `clientObject`-objektet.

Om du redan har en anslutning med det angivna namnet, utlöses ett argumentfelsundantag som anger att anslutningen misslyckades eftersom objektet redan är anslutet.

Varje gång en Flash Player-instans ansluter till den här SWF-filen och försöker att anropa en metod för den angivna lokala anslutningen, skickas begäran till klassen som har angetts med egenskapen `client`, som är inställd på klassen CustomClient1.

```
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");
        }
    }
}
```

För att skapa en LocalConnection-server anropar du metoden `LocalConnection.connect()` och anger sedan ett unikt anslutningsnamn. Om du redan har en anslutning med det angivna namnet, genereras ett ArgumentError-fel som anger att anslutningen misslyckades eftersom objektet redan är anslutet.

Följande utdrag visar hur du skapar en LocalConnection med namnet `conn1`:

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

För att ansluta till det primära programmet från ett sekundärt program måste du först skapa ett LocalConnection-objekt i det sändande LocalConnection-objektet, och sedan anropa metoden `LocalConnection.send()` med namnet på anslutningen och namnet på metoden som ska köras. Om du till exempel vill skicka metoden `doQuit` till det LocalConnection-objekt du skapade tidigare använder du följande kod:

```
sendingConnection.send("conn1", "doQuit");
```

Den här koden ansluter till ett befintligt LocalConnection-objekt med anslutningsnamnet `conn1` och anropar metoden `doMessage()` i fjärrprogrammet. Om du vill skicka parametrar till fjärrprogrammet anger du ytterligare argument efter metodnamnet i metoden `send()` som i följande utdrag:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Ansluta till innehåll i andra domäner och till AIR-program

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att tillåta kommunikation enbart från specifika domäner anropar du metoden `allowDomain()` eller `allowInsecureDomain()` i klassen LocalConnection och anger en lista med en eller flera domäner som ska kunna få tillgång till det här LocalConnection-objektet, och anger på så sätt ett eller flera namn på domäner som ska tillåtas.

I tidigare versioner av ActionScript var `LocalConnection.allowDomain()` och `LocalConnection.allowInsecureDomain()` återkallningsmetoder som utvecklare var tvungna att implementera och dessa var tvungna att returnera ett booleskt värde. I ActionScript 3.0 är `LocalConnection.allowDomain()` och `LocalConnection.allowInsecureDomain()` båda inbyggda metoder som utvecklare kan anropa precis som `Security.allowDomain()` och `Security.allowInsecureDomain()` för att ange ett eller flera namn på domäner som är tillåtna.

I Flash Player 8 används säkerhetsinställningar för lokala SWF-filer. Som standard kan inte en SWF-fil ha åtkomst till både Internet och det lokala filsystemet. Om du anger `localhost` kan valfri lokal SWF-fil få åtkomst till SWF-filen. Om metoden `LocalConnection.send()` försöker att kommunicera med en SWF-fil från en säkerhetssandlåda som den anropande koden inte har tillgång till, skickas en `securityError`-händelse(`SecurityErrorEvent.SECURITY_ERROR`). För att undvika det här felet kan du ange anroparens domän i mottagarens `LocalConnection.allowDomain()`-metod.

Det finns två specialvärden som du kan skicka till metoderna `LocalConnection.allowDomain()` och `LocalConnection.allowInsecureDomain()`: `*` och `localhost`. Asteriskvärdet `(*)` tillåter åtkomst från alla domäner. Strängen `localhost` tillåter anrop till programmet från lokalt installerat innehåll som ligger utanför programresurskatalogen.

Om metoden `LocalConnection.send()` försöker att kommunicera med ett program från en säkerhetssandlåda som den anropande koden inte har tillgång till, skickas en `securityError` -händelse(`SecurityErrorEvent.SECURITY_ERROR`). För att undvika det här felet kan du ange anroparens domän i mottagarens `LocalConnection.allowDomain()`-metod.

Om du implementerar kommunikation enbart mellan innehåll i samma domän, kan du ange en `connectionName`-parameter som inte börjar med understreck (_) och som inte anger något domännamn (till exempel `myDomain:connectionName`). Använd samma sträng i kommandot `LocalConnection.connect(connectionName)`.

Om du implementerar kommunikation mellan innehåll i olika domäner, anger du en `connectionName` -parameter som börjar med understreck. Genom att ange understrecket blir innehållet med det mottagande LocalConnection-objektet mer portabelt mellan domäner. Det här är de två möjliga fallen:

- Om strängen för `connectionName` inte börjar med ett understreck lägger körtiden till ett prefix med huvuddomännamnet och ett kolon (till exempel `myDomain:connectionName`). Detta förhindrar att konflikter uppstår mellan anslutningen och andra anslutningar med samma namn från andra domäner, men alla sändande LocalConnection-objekt måste ange den här huvuddomänen (till exempel `myDomain:connectionName`). Om du flyttar HTML- eller SWF-filen med det mottagande LocalConnection-objektet till en annan domän, ändrar körtiden prefixet så att det motsvarar den nya huvuddomänen (till exempel `anotherDomain:connectionName`). Alla sändande LocalConnection-objekt måste redigeras manuellt så att de pekar på den nya huvuddomänen.
- Om strängen för `connectionName` börjar med ett understreck (till exempel `_connectionName`) lägger körtiden inte till något prefix till strängen. Det innebär att de mottagande och sändande LocalConnection-objekten använder identiska strängar för `connectionName`. Om det mottagande objektet använder `LocalConnection.allowDomain()` för att ange att anslutningar från alla domäner godtas, kan du flytta HTML- eller SWF-filen med det mottagande LocalConnection-objektet till en annan domän utan att ändra några sändande LocalConnection-objekt.

  En nackdel med att använda namn med understreck i `connectionName` är risken för kollisioner, som när två program försöker att ansluta genom att använda samma `connectionName`. En andra, relaterad, nackdel är säkerhetsrelaterad. Anslutningsnamn som använder understreck-syntax identifierar inte det lyssnande programmets domän. Av dessa anledningar är domänkvalificerade namn att föredra.

Adobe AIR

För att kunna kommunicera med innehåll som körs i säkerhetssandlådan för AIR-programmet (innehåll som installerats med AIR-programmet) måste du ge anslutningsnamnet ett prefix med en superdomän som identifierar AIR-programmet. Superdomänsträngen börjar med `app#` följt av program-ID, följt av ett punkttecken (.), följt av utgivar-ID (om det är angivet). Exempelvis är den riktiga superdomänen som ska användas i `connectionName`-parametern för ett program med program-ID `com.example.air.MyApp`, och inget utgivar-ID: `"app#com.example.air.MyApp"`. Om basanslutningsnamnet är "appConnection" är alltså den fullständiga sträng som ska användas i `connectionName`-parametern `"app#com.example.air.MyApp:appConnection"`. Om programmet har utgivar-ID måste även detta ID ingå i superdomänsträngen: `"app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1"`.

När du tillåter att ett annat AIR-program kommunicerar med ditt program genom den lokala anslutningen, måste du anropa `allowDomain()` för LocalConnection-objektet, och skicka med domännamnet för den lokala anslutningen. I ett AIR-program utformas domännamnet utifrån program- och utgivar-ID på samma sätt som anslutningssträngen. Om det sändande AIR-programmet till exempel har ett program-ID som är `com.example.air.FriendlyApp` och ett utgivar-ID som är `214649436BD677B62C33D02233043EA236D13934.1`, blir den domänsträng som du ska använda för att tillåta detta program att ansluta:
`app#com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1`. (Som i AIR 1.5.3 har inte alla AIR-program ett utgivar-ID.)

Kapitel 46: Kommunicera med interna processer i AIR

Adobe AIR 2 och senare

Adobe AIR 2 och senare AIR-program kan köra och kommunicera med andra interna processer via kommandoraden. Ett AIR-program kan till exempel köra en process och samtidigt kommunicera med den via standardströmmarna för indata och utdata.

Om du vill kommunicera med interna processer paketerar du AIR-programmet så att det installeras med ett internt installationsprogram. Det interna installationsprogrammets filtyp är specifik för det operativsystem som det har skapats för:

- I Mac OS är det en DMG-fil.
- I Windows är det en EXE-fil.
- I Linux är det ett RPM- eller DEB-paket.

De här programmen kallas för utökade skrivbordsprofiler. Du skapar en intern installationsfil genom att ange alternativet `-target native` när du anropar kommandot `-package` med hjälp av ADT.

Fler hjälpavsnitt

flash.filesystem.File.openWithDefaultApplication()

flash.desktop.NativeProcess

Översikt över kommunikation med interna processer

Adobe AIR 2 och senare

Ett AIR-program i den utökade skrivbordsprofilen kan köra en fil på samma sätt som om den anropades av kommandoraden. Det kan kommunicera med standardströmmarna i den interna processen. Standardströmmar omfattar strömmar av standardindata (stdin), standardutdata (stdout) och standardfel (stderr).

Obs! *Program i den utökade skrivbordsprofilen kan också starta filer och program med hjälp av metoden* `File.openWithDefaultApplication()`*. Men när den här metoden används får inte AIR-programmet åtkomst till standardströmmarna. Mer information finns i* *"Öppna filer med standardsystemprogrammet" på sidan 683*

Följande kodexempel visar hur programmet test.exe startas i programkatalogen. Programmet skickar argumentet `"hello"` som ett kommandoradsargument och lägger till en händelseavlyssnare i processens standardutdataström:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs.push("hello");
nativeProcessStartupInfo.arguments = processArgs;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, onOutputData);
process.start(nativeProcessStartupInfo);
public function onOutputData(event:ProgressEvent):void
{
    var stdOut:ByteArray = process.standardOutput;
    var data:String = stdOut.readUTFBytes(process.standardOutput.bytesAvailable);
    trace("Got: ", data);
}
```

Starta och avsluta en intern process

Adobe AIR 2 och senare

Om du vill starta en intern process anger du att objektet NativeProcessInfo ska göra följande:

- Peka på den fil som du vill starta
- Spara kommandoradsargument som ska skickas till processen när den startas (valfritt)
- Ange processens arbetskatalog (valfritt)

Om du vill starta den interna processen skickar du objektet NativeProcessInfo som parametern för metoden `start()` för ett NativeProcess-objekt.

Följande kodexempel visar till exempel hur programmet test.exe startas i programkatalogen. Programmet skickar argumentet `"hello"` och anger användarens dokumentkatalog som arbetskatalog:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs[0] = "hello";
nativeProcessStartupInfo.arguments = processArgs;
nativeProcessStartupInfo.workingDirectory = File.documentsDirectory;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
```

Om du vill avsluta processen anropar du NativeProcess-objektets `exit()`-metod

Om du vill att en fil ska kunna köras i det installerade programmet bör du kontrollera att den är körbar på filsystemet när du packar programmet. (På Mac och Linux kan du använda chmod för att ange den körbara flaggan, om detta behövs.)

Kommunicera med en intern process

Adobe AIR 2 och senare

När ett AIR-program har startat en intern process kan det kommunicera med processens strömmar av standardindata, standardutdata och standardfel.

Du skriver data till och läser data från strömmarna med hjälp av följande egenskaper i objektet NativeProcess:

- `standardInput` – Innehåller åtkomst till data för standardindataströmmen.
- `standardOutput` – Innehåller åtkomst till data för standardutdataströmmen.
- `standardError` – Innehåller åtkomst till data för standardfelströmmen.

Skriva till standardindataströmmen

Du kan skriva data till standardindataströmmen med hjälp av skrivmetoder för egenskapen `standardInput` för objektet NativeProcess. När AIR-programmet skriver data till processen skickar objektet NativeProcess `standardInputProgress`-händelser.

Om ett fel inträffar när data skrivs till standardindataströmmen skickar objektet NativeProcess en `ioErrorStandardInput`-händelse.

Du kan stänga indataströmmen genom att anropa metoden `closeInput()` i objektet NativeProcess. När indataströmmen stängs skickar objektet NativeProcess en `standardInputClose`-händelse.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
process.standardInput.writeUTF("foo");
if(process.running)
{
    process.closeInput();
}
```

Läsa från standardutdataströmmen

Du kan läsa data från standardutdataströmmen med hjälp av den här egenskapens läsmetoder. När AIR-programmet tar emot data från processens utdataström skickar objektet NativeProcess `standardOutputData`-händelser.

Om ett fel inträffar när data skrivs till standardutdataströmmen skickar objektet NativeProcess en `standardOutputError`-händelse.

När processen avslutar utdataströmmen skickar objektet NativeProcess en `standardOutputClose`-händelse.

Se till att data läses samtidigt som de genereras när de läses från standardindataströmmen. Lägg med andra ord till en händelseavlyssnare för händelsen `standardOutputData`. Läs data i händelseavlyssnaren `standardOutputData` från egenskapen `standardOutput` för objektet NativeProcess. Vänta inte bara på att händelsen `standardOutputClose` eller `exit` ska läsa alla data. Om du inte läser data samtidigt som de genereras av den interna processen kan bufferten bli full och data kan förloras. En full buffert kan göra så att den interna processen hänger sig när mer data ska skrivas. Men om du inte registrerar en händelseavlyssnare för händelsen `standardOutputData` kommer bufferten inte att bli full och processen kommer inte att hänga sig. I det här fallet har du ingen åtkomst till dessa data.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, dataHandler);
process.start(nativeProcessStartupInfo);
var bytes:ByteArray = new ByteArray();
function dataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardOutput.readBytes(process.standardOutput.bytesAvailable);
}
```

Läsa från standardfelströmmen

Du kan läsa data från standardfelströmmen med hjälp av den här egenskapens läsmetoder. När AIR-programmet läser felströmsdata från processen skickar objektet NativeProcess `standardErrorData`-händelser.

Om ett fel inträffar när data skrivs till standardfelströmmen skickar objektet NativeProcess en `standardErrorIoError`-händelse.

När processen stänger felströmmen skickar objektet NativeProcess en `standardErrorClose`-händelse..

Se till att data läses samtidigt som de genereras när de läses från standardfelströmmen. Lägg med andra ord till en händelseavlyssnare för händelsen `standardErrorData`. Läs data i händelseavlyssnaren `standardErrorData` från egenskapen `standardError` för objektet NativeProcess. Vänta inte bara på att händelsen `standardErrorClose` eller `exit` ska läsa alla data. Om du inte läser data samtidigt som de genereras av den interna processen kan bufferten bli full och data kan förloras. En full buffert kan göra så att den interna processen hänger sig när mer data ska skrivas. Men om du inte registrerar en händelseavlyssnare för händelsen `standardErrorData` kommer bufferten inte att bli full och processen kommer inte att hänga sig. I det här fallet har du ingen åtkomst till dessa data.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_ERROR_DATA, errorDataHandler);
process.start(nativeProcessStartupInfo);
var errorBytes:ByteArray = new ByteArray();
function errorDataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardError.readBytes(process.standardError.bytesAvailable);
}
```

Säkerhetsaspekter för kommunikation med interna processer

Adobe AIR 2 och senare

Den interna processen API kan köra alla körbara filer på användarens system. Du bör vara mycket försiktig när du konstruerar och kör kommandon. Om någon del av ett kommando som ska köras kommer från en extern källa kontrollerar du noga att det kommandot är säkert att köra. På samma sätt bör AIR-programmet validera alla data som skickas till en process som körs.

Det kan dock vara svårt att verifiera indata. För att slippa den här typen av problem är det bäst att skriva ett inbyggt program (t.ex. en EXE-fil i Windows) med specifika API:er. Dessa API:er bör endast bearbeta de kommandon som definieras av programmet. Exempelvis kanske programmet endast accepterar en begränsad uppsättning instruktioner via standardindataströmmen.

Med AIR på Windows kan du inte köra .bat-filer direkt. Kommandotolken (cmd.exe) kör .bat-filer i Windows. När du anropar en .bat-fil kan den här kommandotolken tolka de argument som skickas till kommandot som fler program som ska startas. Skadliga extratecken i argumentsträngen kan medföra att cmd.exe kör ett skadligt eller osäkert program. Utan korrekt dataverifiering kan ditt AIR-program till exempel anropa `myBat.bat myArguments c:/evil.exe`. Kommandotolken skulle då starta programmet evil.exe utöver din .bat-fil.

Kapitel 47: Använda externt API

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Den externa API:n i ActionScript 3.0 (flash.external.ExternalInterface) möjliggör enkel kommunikation mellan ActionScript och behållarprogrammet som Adobe Flash Player körs i. Använd ExternalInterface-API:t för att skapa interaktion mellan ett SWF-dokument och JavaScript på en HTML-sida.

Du kan inte använda det externa API:t för att interagera med ett behållarprogram, skicka data mellan ActionScript och JavaScript på en HTML-sida.

Några vanliga externa API-uppgifter är:

- Hämta information om behållarprogrammet
- Använda ActionScript för att anropa kod på en webbsida som visas i en webbläsare eller i ett AIR-program
- Anropa ActionScript-kod från en webbsida
- Skapa en proxy som förenklar anrop till ActionScript-kod från en webbsida

Obs! *Här beskrivs endast kommunikationen mellan ActionScript i en SWF-fil och behållarprogrammet som innehåller en referens till Flash Player-instansen till vilken SWF-filen läses in. Annan användning av Flash Player i ett program ligger utanför omfånget av den här dokumentationen. Flash Player är utformat för att användas som ett webbläsarplugin-program eller som en projektor (fristående program). Andra användarscenarier kanske har begränsat stöd.*

Använda externt API i AIR

Eftersom AIR-program inte har någon extern behållare gäller vanligtvis inte det här externa gränssnittet och behövs inte heller. När AIR-programmet läser in en SWF-fil direkt kan programkoden kommunicera direkt med ActionScript-koden i SWF-filen (påverkas av begränsningar från säkerhetssandlådor).

Men om AIR-programmet läser in en SWF-fil med en HTML-sida i ett HTMLLoader-objekt (eller en HTML-komponent i Flex) fungerar HTMLLoader-objektet som den externa behållaren. Du kan därför använda det externa gränssnittet för att kommunicera mellan ActionScript-koden i den inlästa SWF-filen och JavaScript-koden i HTML-sidan som lästes in i HTMLLoader.

Grunderna i externt API

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Även om en SWF-fil i vissa fall kan köras fristående (till exempel om du använder Adobe® Flash® Professional för att skapa en SWF-projektor) körs ett SWF-program i de flesta fall som ett element i ett annat program. Oftast är behållaren som innehåller SWF-filen en HTML-fil. Mindre vanligt är att SWF-filen används som ett helt eller en del av ett användargränssnitt i ett skrivbordsprogram.

När du arbetar med mer avancerade program kan du upptäcka att du behöver upprätta kommunikation mellan SWF-filen och behållarprogrammet. Det är till exempel vanligt att en webbsida visar text eller annan information i HTML-format och att en SWF-fil används för att visa dynamiskt visuellt innehåll som diagram eller video. I så fall kanske du vill att något ska ändras i SWF-filen när användaren klickar på en knapp på webbsidan. I ActionScript finns en mekanism som kallas det externa API:t (gränssnittet) som underlättar sådan kommunikation mellan ActionScript i en SWF-fil och annan kod i behållarprogrammet.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer om den här funktionen:

Behållarprogram Programmet där Flash Player kör en SWF-fil, till exempel en webbläsare eller en HTML-sida som har Flash Player-innehåll eller ett AIR-program som läser in SWF-filen i en webbsida.

Projektor En körbar fil med SWF-innehåll och en inbäddad version av Flash Player. Du kan skapa en projektorfil med Flash Professional eller den fristående Flash Player. Projektorer används ofta för att distribuera SWF-filer på CD-ROM-skivor eller i andra fall där filstorleken inte är ett hinder och utvecklaren vill vara säker på att alla användare kan köra SWF-filen, oavsett vilken version av Flash Player som är installerad på användarens dator.

Proxy Ett mellanprogram eller en mellankod som anropar kod i ett program (det ”externa programmet”) för ett annat programs räkning (det ”anropande programmet”) och returnerar värden till det anropande programmet. En proxy kan användas av olika skäl, bland annat:

- För att förenkla processen att göra externa funktionsanrop genom att konvertera inbyggda funktionsanrop i det anropande programmet till ett format som det externa programmet förstår.
- För att förbigå säkerhetsbegränsningar eller andra hinder som gör att anroparen inte kan kommunicera direkt med det externa programmet.

Serialisera Att konvertera objekt eller datavärden till ett format som kan användas för att skicka värdena i meddelanden mellan två programmeringssystem, till exempel via Internet eller mellan två olika program som körs på en enda dator.

Arbeta med exemplen

Många av kodexemplen här är små, demonstrativa kodavsnitt, inte kompletta fungerande exempel eller kod som kontrollerar värden. Eftersom det externa API:t (per definition) kräver att du skriver både ActionScript-kod och kod i ett behållarprogram måste en behållare (till exempel en webbsida som innehåller SWF-filen) skapas och koden måste interagera med behållaren för att exemplen ska kunna testas.

Så här testar du ett exempel på kommunikation mellan ActionScript och JavaScript:

1. Skapa ett nytt dokument med Flash Professional och spara det på datorn.
2. Välj Arkiv > Publiceringsinställningar på huvudmenyn.
3. Kontrollera att kryssrutorna Flash och HTML är markerade på fliken Format i dialogrutan Publiceringsinställningar.
4. Klicka på knappen Publicera. Då skapas en SWF-fil och en HTML-fil i samma mapp och med samma namn som du använde för att spara dokumentet. Klicka på OK för att stänga dialogrutan Publiceringsinställningar.
5. Avmarkera kryssrutan HTML. Nu när HTML-sidan har skapats ska du ändra den till JavaScript-kod. När du har avmarkerat kryssrutan HTML kommer inte Flash att skriva över ändringarna med en ny HTML-sida när du har ändrat HTML-sidan och Flash publicerar SWF-filen.
6. Klicka på OK för att stänga dialogrutan Publiceringsinställningar.

7 Öppna HTML-filen som skapades av Flash när du publicerade SWF-filen med en HTML- eller textredigerare. Lägg till `script`-taggar för öppning och stängning, och kopiera dem till JavaScript-koden från kodexemplen:

```
<script>
// add the sample JavaScript code here
</script>
```

8 Spara HTML och gå tillbaka till Flash.

9 Markera nyckelbildrutan i ram 1 på tidspanelen och öppna sedan åtgärdspanelen.

10 Kopiera ActionScript-kodexemplet till rutan Skript.

11 Välj Arkiv > Publicera på huvudmenyn så att SWF-filen uppdateras med ändringarna du gjort.

12 Öppna HTML-sidan du redigerade med en webbläsare så att du kan se sidan och testa kommunikationen mellan ActionScript och HTML-sidan.

Så här testar du ett exempel på kommunikation mellan ActionScript och en ActiveX-behållare:

1 Skapa ett nytt dokument med Flash Professional och spara det på datorn. Det kan vara praktiskt att spara det i mappen där behållarprogrammet kommer att förvänta sig att hitta SWF-filen.

2 Välj Arkiv > Publiceringsinställningar på huvudmenyn.

3 Kontrollera att bara kryssrutan Flash är markerad på fliken Format i dialogrutan Publiceringsinställningar.

4 Klicka på mappikonen i fältet Arkiv bredvid kryssrutan Flash och markera den mapp som SWF-filen ska publiceras till. Genom att ange platsen för SWF-filen kan du (till exempel) ha dokumentet i en mapp, men lägga den publicerade SWF-filen i en annan mapp, till exempel den mapp som innehåller källkoden för behållarprogrammet.

5 Markera nyckelbildrutan i ram 1 på tidspanelen och öppna sedan åtgärdspanelen.

6 Kopiera ActionScript-koden för exemplet till rutan Skript.

7 Välj Arkiv > Publicera på huvudmenyn när du vill publicera om SWF-filen.

8 Skapa och kör behållarprogrammet för att testa kommunikationen mellan ActionScript och behållarprogrammet.

Ett komplett exempel om hur du använder ett externt API för att kommunicera med en HTML-sida finns under följande ämne:

- ”Exempel på extern API: kommunicera mellan ActionScript och JavaScript i en webbläsare” på sidan 851

Exemplen innehåller den fullständiga koden, inklusive felsökningskod för ActionScript och behållarprogrammet, som bör användas när du skriver kod med det externa API:t. Ytterligare ett fullständigt exempel på hur du använder det externa API:t finns i klassexemplet för klassen ExternalInterface i Referens för ActionScript 3.0.

Krav och fördelar för det externa API:t

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det externa API:t är den del av ActionScript som innehåller mekanismen för kommunikation mellan ActionScript och kod som körs i ett externt program som fungerar som behållare för Flash Player (oftast en webbläsare eller ett fristående projektorprogram). I ActionScript 3.0 finns det externa API:ts funktionalitet i klassen ExternalInterface. I Flash Player-versioner före Flash Player 8 användes åtgärden `fscommand()` för att utföra kommunikation med behållarprogrammet. Klassen ExternalInterface ersätter `fscommand()`.

Obs! *Om du behöver använda den gamla funktionen* `fscommand()`*, till exempel för att bibehålla kompatibiliteten med äldre program eller för att interagera med ett SWF-behållarprogram från tredje part eller den fristående Flash Player, finns den fortfarande som en funktion på paketnivå i paketet flash.system.*

Klassen ExternalInterface är ett undersystem som gör att du enkelt kan kommunicera från ActionScript och Flash Player till JavaScript på en HTML-sida.

Klassen ExternalInterface är bara tillgänglig under följande omständigheter:

- I alla användbara versioner av Internet Explorer för Windows (5.0 och senare)
- I valfri webbläsare som stöder NPRuntime-gränssnittet, d.v.s. Firefox 1.0 och senare, Mozilla 1.7.5 och senare, Netscape 8.0 och senare samt Safari 1.3 och senare.
- I ett AIR-program när SWF-filen är inbäddad i en HTML-sida som visas av HTMLLoader-kontrollen.

I alla andra lägen (till exempel när programmet körs i en fristående spelare) returnerar egenskapen `ExternalInterface.availablefalse`.

Från ActionScript kan du anropa en JavaScript-funktion på HTML-sidan. Externt API erbjuder följande förbättrade funktioner jämfört med `fscommand()`:

- Du kan använda alla JavaScript-funktioner, inte bara de funktioner som du kan använda med `fscommand()`.
- Du kan överföra valfritt antal argument, med valfria namn. Du begränsas inte till att överföra ett kommando och ett argument med en enda sträng. Det ger externt API mycket större flexibilitet än `fscommand()`.
- Du kan skicka olika datatyper (till exempel Boolean, Number och String). Du begränsas inte till String-parametrar.
- Du kan ta emot värdet för ett anrop och detta värde returneras omedelbart till ActionScript (som returvärde för anropet du gör).

Viktigt! *Om det namn som ges Flash Player-instansen på en HTML-sida (*`id`*-attributet för taggen* `object`*) innehåller ett bindestreck (*`-`*) eller andra tecken som definieras som operatorer i JavaScript (till exempel* `+`*,* `*`*,* `/`*,* `\`*,* `.`*) fungerar ExternalInterface-anrop från ActionScript inte när behållarwebbsidan visas i Internet Explorer. Dessutom fungerar ExternalInterface-anrop från ActionScript inte heller om de HTML-taggar som definierar Flash Player-instansen (taggarna* `object` *och* `embed`*) är kapslade i en HTML* `form`*-tagg.*

Använda klassen ExternalInterface

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Kommunikation mellan ActionScript och behållarprogrammet kan ha en av två former: antingen kan ActionScript anropa kod (till exempel en JavaScript-funktion) som definieras i behållaren, eller också kan kod i behållaren anropa en ActionScript-funktion som har definierats som anropbar. I båda fallen kan information skickas till koden som anropas och resultatet kan returneras till koden som gör anropet.

Klassen ExternalInterface har två statiska egenskaper och två statiska metoder som underlättar kommunikationen. Egenskaperna och metoderna används för att hämta information om den externa gränssnittsanslutningen, för att köra kod i behållaren från ActionScript och för att göra ActionScript-funktioner tillgängliga för anrop från behållaren.

Hämta information om den externa behållaren

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Egenskapen `ExternalInterface.available` anger om Flash Player är i en behållare som har ett externt gränssnitt. Om det externa gränssnittet är tillgängligt är egenskapen `true`. Annars är den `false`. Innan du använder några andra funktioner i klassen ExternalInterface måste du kontrollera att den aktuella behållaren stöder extern gränssnittskommunikation på följande sätt:

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

Obs! *Egenskapen `ExternalInterface.available` rapporterar om den aktuella behållaren är en typ som stöder ExternalInterface-anslutbarhet. Den talar inte om huruvida JavaScript är aktiverat i den webbläsare som används.*

Med egenskapen `ExternalInterface.objectID` kan du avgöra Flash Player-instansens unika identifierare (`id`-attributet för `object`-taggen i Internet Explorer eller `name`-attributet för `embed`-taggen i webbläsare som använder NPRuntime-gränssnittet). Detta unika ID representerar det SWF-dokument som är i webbläsaren och kan användas för att referera till SWF-dokumentet, till exempel vid anrop till en JavaScript-funktion i en HTML-behållarsida. När Flash Player-behållaren inte är en webbsida är egenskapen `null`.

Anropa extern kod från ActionScript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `ExternalInterface.call()` kör kod i behållarprogrammet. Den måste ha minst en parameter, en sträng som innehåller namnet på den funktion som ska anropas i behållarprogrammet. Eventuella ytterligare parametrar som skickas till metoden `ExternalInterface.call()` skickas vidare till behållaren som parametrar för funktionsanropet.

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Om behållaren är en HTML-sida aktiverar metoden JavaScript-funktionen med det angivna namnet, som måste definieras i ett `script`-element i HTML-behållarsidan. Returvärdet för JavaScript-funktionen skickas tillbaka till ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Om behållaren är någon annan ActiveX-behållare gör metoden att Flash Player ActiveX-kontrollen skickar händelsen `FlashCall`. Det angivna funktionsnamnet och eventuella parametrar serialiseras till en XML-sträng av Flash Player. Behållaren har åtkomst till informationen i egenskapen `request` för händelseobjektet och kan använda den för att avgöra hur den egna koden ska köras. För att returnera ett värde till ActionScript anropar behållarens kod ActiveX-objektets `SetReturnValue()`-metod och skickar resultatet (serialiserat till en XML-sträng) som en parameter för den metoden. Mer information om det XML-format som används för den här kommunikationen finns i "Det externa API:ts XML-format" på sidan 850.

Oavsett om behållaren är en webbläsare eller en annan ActiveX-behållare returneras `null` om anropet misslyckas eller om behållarens metod inte anger ett returvärde. Metoden `ExternalInterface.call()` ger ett SecurityError-undantag om behållarmiljön tillhör en säkerhetssandlåda som den anropande koden inte har tillgång till. Du kan undvika detta genom att ställa in ett lämpligt värde för `allowScriptAccess` i behållarmiljön. För att ändra värdet för `allowScriptAccess` i en HTML-sida redigerar du motsvarande attribut i taggarna `object` och `embed`.

Anropa ActionScript-kod från behållaren

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En behållare kan bara anropa ActionScript-kod som är i en funktion. Ingen annan ActionScript-kod kan anropas av en behållare. För att kunna anropa en ActionScript-funktion från behållarprogrammet måste du göra två saker: registrera funktionen med klassen ExternalInterface och sedan anropa den från behållarens kod.

Först måste du registrera ActionScript-funktionen för att ange att den ska vara tillgänglig för behållaren. Använd metoden `ExternalInterface.addCallback()` på följande sätt:

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

Metoden `addCallback()` används med två parametrar. Den första, som är ett funktionsnamn i form av en sträng, är namnet som kommer att användas för funktionen i behållaren. Den andra parametern är den faktiska ActionScript-funktion som körs när behållaren anropar det definierade funktionsnamnet. Eftersom namnen är åtskilda kan du ange ett funktionsnamn som ska användas i behållaren även om ActionScript-funktionen har ett annat namn. Det är särskilt användbart om funktionsnamnet är okänt, till exempel om en anonym funktion anges eller om det inte avgörs förrän under körning vilken funktion som ska anropas.

När en ActionScript-funktion har registrerats i klassen ExternalInterface kan behållaren anropa den. Hur detta görs varierar beroende på behållartypen. I JavaScript-kod i en webbläsare anropas till exempel ActionScript-funktionen med det registrerade funktionsnamnet som om det vore en metod i Flash Player-objektet i webbläsaren (d.v.s. en metod i JavaScript-objektet som representerar `object`- eller `embed`-taggen). Med andra ord skickas parametrar och resultatet returneras som om en lokal funktion hade anropats.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

När en ActionScript-funktion anropas i en SWF-fil som körs i ett skrivbordsprogram måste det registrerade funktionsnamnet och alla parametrar serialiseras i en XML-formaterad sträng. Anropet utförs sedan genom att metoden `CallFunction()` i ActiveX-kontrollen anropas med XML-strängen som parameter. Mer information om det XML-format som används för den här kommunikationen finns i "Det externa API:ts XML-format" på sidan 850.

I båda fallen skickas ActionScript-funktionens returvärde tillbaka till behållarkoden, antingen direkt som ett värde när anroparen är JavaScript-kod i en webbläsare eller serialiserad som en XML-formaterad sträng när anroparen är en ActiveX-behållare.

Det externa API:ts XML-format

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För kommunikation mellan ActionScript och ett program med Shockwave Flash ActiveX-kontrollen används ett specifikt XML-format för att koda funktionsanrop och värden. XML-formatet som används av det externa API:t har två delar. Ett format används för att representera funktionsanrop. Ett annat format används för att representera enskilda värden. Det formatet används både för parametrar i funktioner och för funktionernas returvärden. XML-formatet för funktionsanrop används för anrop till och från ActionScript. För ett funktionsanrop från ActionScript skickar Flash Player XML-strängen till behållaren. För ett anrop från behållaren förväntar sig Flash Player att behållarprogrammet ska skicka en XML-sträng i det här formatet. I följande XML-fragment visas ett exempel på ett XML-formaterat funktionsanrop:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

Rotnoden kallas `invoke`-nod. Den har två attribut: `name` anger namnet på funktionen som ska anropas och `returntype` är alltid `xml`. Om funktionsanropet innehåller parametrar har `invoke`-noden en underordnad `arguments`-nod vars underordnade noder är de parametervärden som formaterats med det enskilda värdeformatet som förklaras härnäst.

För enskilda värden, inklusive funktionens parametrar och funktionens returvärden, använder du ett formateringsschema som förutom de faktiska värdena innehåller datatypsinformation. I följande tabell listas ActionScript-klasser och det XML-format som används för att koda värden som tillhör den datatypen:

klassen ActionScript/värde	klassen C#/värde	Format	Kommentarer
null	null	<null/>	
Boolean `true`	bool true	<true/>	
Boolean `false`	bool false	<false/>	
String	string	<string>strängvärde</string>	
Number, int, uint	single, double, int, uint	`<number>27.5</number>` `<number>-12</number>`	

klassen ActionScript/värde	klassen C#/värde	Format	Kommentarer

Array (elementen kan vara av olika typer)	En samling som tillåter element av olika typer, till exempel ArrayList eller object[]	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			Noden `property` definierar enskilda element och attributet `id` är det numeriska, nollbaserade indexvärdet.

Object	En ordlista med strängnycklar och objektvärden, till exempel en HashTable med strängnycklar	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			Noden `property` definierar enskilda egenskaper och attributet `id` är egenskapsnamnet (en sträng).

Andra inbyggda eller anpassade klasser		`<null/> or` `<object></object>`	ActionScript kodar andra objekt som null eller som tomma objekt. Oavsett vilket försvinner eventuella egenskapsvärden.

Obs! *Den här tabellen visar ActionScript-klasser och motsvarande C#-klasser som exempel. Det externa API:t kan emellertid användas för att kommunicera med alla programmeringsspråk eller körningsmiljöer som stöder ActiveX-kontroller, inte bara C#-program.*

När du skapar egna program med det externa API:t med ett ActiveX-behållarprogram är det antagligen praktiskt att skriva en proxy som utför konverteringen av programspecifika funktionsanrop till det serialiserade XML-formatet. Ett exempel på en proxyklass skriven i C# finns i Inuti klassen ExternalInterfaceProxy.

Exempel på extern API: kommunicera mellan ActionScript och JavaScript i en webbläsare

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I det här exempelprogrammet demonstreras lämpliga tekniker för kommunikation mellan ActionScript och JavaScript i en webbläsare, i det här fallet i ett snabbmeddelandeprogram som gör att en person kan chatta med sig själv (därav programnamnet: Introvert IM). Meddelanden skickas mellan ett HTML-formulär på webbsidan och ett SWF-gränssnitt med det externa API:t. Teknikerna som demonstreras i det här exemplet innehåller följande:

- Initiera kommunikationen korrekt genom att verifiera att webbläsaren är klar att kommunicera innan kommunikationen startar
- Kontrollera om behållaren stöder det externa API:t
- Anropa JavaScript-funktioner från ActionScript, skicka parametrar och ta emot värden som svar
- Göra ActionScript-metoder tillgängliga för anrop från JavaScript och utföra anropen

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Programfilerna för Introvert IM finns i mappen Samples/IntrovertIM_HTML. Programmet består av följande filer:

Fil	Beskrivning
IntrovertIMApp.fla eller IntrovertIMApp.mxml	Huvudprogramfilen för Flash (FLA) eller Flex (MXML).
com/example/programmingas3/introvertIM/IMManager.as	Klassen som upprättar och hanterar kommunikationen mellan ActionScript och behållaren.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Anpassad händelsetyp som skickas av klassen IMManager när ett meddelande tas emot från behållaren.
com/example/programmingas3/introvertIM/IMStatus.as	En uppräkning vars värden representerar de olika värdena för tillgänglighetsstatus som kan väljas i programmet.
html-flash/IntrovertIMApp.html eller html-template/index.template.html	HTML-sidan för programmet för Flash (html-flash/IntrovertIMApp.html) eller den mall som används för att skapa HTML-behållarsidan för programmet för Adobe Flex (html-template/intex.template.html). Filen innehåller alla de JavaScript-funktioner som programmets behållardel består av.

Förberedelser för kommunikation mellan ActionScript och webbläsaren

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Ett av de vanligaste sätten att använda det externa API:t är för att låta ActionScript-program kommunicera med en webbläsare. Med det externa API:t kan ActionScript-metoder anropa kod som skrivits i JavaScript och vice versa. Eftersom webbläsare är så komplicerade och eftersom de renderar sidor internt går det inte att garantera att ett SWF-dokument kan registrera sina callback-funktioner innan den första JavaScripten på HTML-sidan körs. Därför bör alltid SWF-dokumentet anropa HTML-sidan för att meddela den att SWF-dokumentet är klart att ta emot anslutningar innan funktionerna i SWF-dokumentet anropas från JavaScript.

Introvert IM avgör till exempel genom en serie steg som utförs av klassen IMManager om webbläsaren är klar för kommunikation och förbereder SWF-filen för kommunikation. Det första steget, som avgör när webbläsaren är klar för kommunikation, utförs i IMManager-konstruktorn på följande sätt:

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

Först kontrolleras om det externa API:t överhuvudtaget är tillgängligt i den aktuella behållaren med hjälp av egenskapen `ExternalInterface.available`. I så fall börjar kommunikationen upprättas. På grund av att säkerhetsundantag och andra fel kan uppstå när du försöker kommunicera med ett externt program är koden placerad i ett `try`-block (motsvarande `catch`-block har utelämnats i exemplet av utrymmesskäl).

Sedan anropas metoden `isContainerReady()` som listas här:

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

Metoden `isContainerReady()` använder i sin tur metoden `ExternalInterface.call()` för att anropa JavaScript-funktionen `isReady()` på följande sätt:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

Funktionen `isReady()` returnerar helt enkelt värdet för variabeln `jsReady`. Den variabeln är från början `false`. När webbsidans `onload`-händelse aktiveras ändras variabelns värde till `true`. Med andra ord svarar JavaScript `false` på anropet `ExternalInterface.call("isReady")` om ActionScript anropar funktionen `isReady()` innan sidan har lästs in. Det gör i sin tur att ActionScript-metoden `isContainerReady()` returnerar `false`. När sidan har lästs in returnerar JavaScript-funktionen `isReady()true` så att även ActionScript-metoden `isContainerReady()` returnerar `true`.

I IMManager-konstruktorn händer en av två saker beroende på om behållaren är klar eller inte. Om `isContainerReady()` returnerar `true` anropar koden helt enkelt metoden `setupCallbacks()` som slutför processen att upprätta kommunikation med JavaScript. Om `isContainerReady()` å andra sidan returnerar `false` pausas i stället processen. Ett Timer-objekt skapas och instrueras att anropa metoden `timerHandler()` var hundrade millisekund på följande sätt:

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

Varje gång metoden `timerHandler()` anropas kontrollerar den resultatet av metoden `isContainerReady()`. När behållaren har initierats returnerar metoden `true`. Sedan stoppar koden Timer-objektet och anropar metoden `setupCallbacks()` som slutför processen att upprätta kommunikation med webbläsaren.

Exponera ActionScript-metoder för JavaScript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Föregående exempel visade, att när koden avgör att webbläsaren är klar, anropas metoden `setupCallbacks()`. Denna metod förbereder ActionScript för att ta emot anrop från JavaScript, vilket visas nedan:

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

Metoden `setCallBacks()` avslutar förberedelserna för kommunikation med behållaren genom att anropa `ExternalInterface.addCallback()` för att registrera de två metoder som ska vara tillgängliga för anrop från JavaScript. I den här koden är den första parametern – det namn som metoden är känd under i JavaScript (`"newMessage"` och `"getStatus"`) – samma som metodens namn i ActionScript. (I det här fallet fanns det ingen fördel med att använda olika namn. Därför användes samma namn för enkelhetens skull.) Slutligen används metoden `ExternalInterface.call()` för att anropa JavaScript-funktionen `setSWFIsReady()`, som meddelar behållaren att ActionScript-funktionerna har registrerats.

Kommunikation från ActionScript till webbläsaren

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Programmet Introvert IM demonstrerar ett antal exempel på hur JavaScript-funktioner anropas i behållarsidan. I det enklaste fallet (ett exempel från metoden `setupCallbacks()`) anropas JavaScript-funktionen `setSWFIsReady()` utan att några parametrar skickas och utan att något returvärde tas emot:

```
ExternalInterface.call("setSWFIsReady");
```

I ett annat exempel från metoden `isContainerReady()` anropar ActionScript funktionen `isReady()` och tar emot ett booleskt värde som svar:

```
var result:Boolean = ExternalInterface.call("isReady");
```

Du kan också skicka parametrar till JavaScript-funktioner med det externa API:t. Ta till exempel metoden `sendMessage()` i klassen IMManager. Den anropas när användaren skickar ett nytt meddelande till sin ”konversationspartner”:

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

Återigen är det `ExternalInterface.call()` som används för att anropa den angivna JavaScript-funktionen och meddela webbläsaren om det nya meddelandet. Dessutom skickas själva meddelandet som en extra parameter till `ExternalInterface.call()`. Därmed skickas det även som en parameter till JavaScript-funktionen `newMessage()`.

Anropa ActionScript-kod från JavaScript

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Kommunikation ska ske åt två håll. Programmet Introvert IM är inget undantag. Det är inte bara Flash Players IM-klient som anropar JavaScript för att skicka meddelanden, HTML-formuläret anropar också JavaScript-koden för att skicka meddelanden till och få information från SWF-filen. När SWF-filen till exempel meddelar behållaren att den har upprättat kontakt och är klar att kommunicera, är det första webbläsaren gör att anropa metoden `getStatus()` i klassen IMManager för att hämta den ursprungliga statusen för användartillgänglighet från SWF-filens IM-klient. Detta görs i webbsidan med funktionen `updateStatus()` på följande sätt:

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

Koden kontrollerar värdet för variabeln `swfReady`, som håller reda på om SWF-filen har meddelat webbläsaren att den har registrerat sina metoder med klassen ExternalInterface. Om SWF-filen är klar att ta emot kommunikation anropar nästa rad (`var currentStatus =` ...) metoden `getStatus()` i klassen IMManager. Tre saker händer på den här kodraden:

- JavaScript-funktionen `getSWF()` anropas och returnerar en referens till det JavaScript-objekt som representerar SWF-filen. Den parameter som skickas till `getSWF()` avgör vilket webbläsarobjekt som returneras om det finns mer än en SWF-fil i en HTML-sida. Det värde som skickas till den parametern måste matcha attributet `id` i `object`-taggen och attributet `name` i `embed`-taggen som används för att inkludera SWF-filen.
- Med hjälp av referensen till SWF-filen anropas metoden `getStatus()` som om den vore en metod i SWF-objektet. I det här fallet används funktionsnamnet ”`getStatus`” eftersom det är namnet som ActionScript-funktionen är registrerad under med `ExternalInterface.addCallback()`.
- ActionScript-metoden `getStatus()` returnerar ett värde och det värdet tilldelas variabeln `currentStatus`, som sedan tilldelas som innehåll (egenskapen `value`) för textfältet `status`.

Obs! *Om du följer koden märker du förmodligen att i källkoden för funktionen `updateStatus()` har kodraden som anropar funktionen `getSWF()` skrivits så här: var currentStatus = getSWF("${application}").getStatus(); Texten `${application}` är en platshållare i HTML-sidmallen. När Adobe Flash Builder genererar den faktiska HTML-sidan för programmet, ersätts den här platshållaren med samma text som används som `object`-taggens `id`-attribut och `embed`-taggens `name`-attribut (`IntrovertIMApp` i exemplet). Det är värdet som förväntas av funktionen `getSWF()`.*

JavaScript-funktionen `sendMessage()` demonstrerar hur en parameter skickas till en ActionScript-funktion. (`sendMessage()` är den funktion som anropas när användaren trycker på knappen Send på HTML-sidan.)

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

ActionScript-metoden `newMessage()` förväntar sig en parameter, så JavaScript-variabeln `message` skickas till ActionScript genom att användas som parameter i anropet till metoden `newMessage()` i JavaScript-koden.

Identifiera webbläsartypen

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom webbläsare öppnar innehåll på olika sätt är det viktigt att alltid använda JavaScript för att identifiera vilken webbläsare användaren har och att öppna filmen i enlighet med den webbläsarspecifika syntaxen, med objektet window eller document, vilket visas i JavaScript-funktionen `getSWF()` i det här exemplet:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Om skriptet inte kan identifiera användarens webbläsartyp kan användaren se oväntade effekter när SWF-filer spelas upp i en HTML-behållare.

Kapitel 48: XML-signaturvalidering i AIR

Adobe AIR 1.5 och senare

Använd klasserna i XMLSignatureValidator-API:t i Adobe® AIR® om du vill validera digitala signaturer i enlighet med en deluppsättning av W3C:s rekommendation för XML-signatursyntax och -bearbetning (http://http://www.w3.org/TR/xmldsig-core/). XML-signaturer kan användas för att verifiera dataintegriteten och signerarens identitet.

XML-signaturer kan användas för att validera meddelanden eller resurser som laddats ned med programmet. Om programmet t. ex. innehåller abonnerade tjänster kan du inkludera abonnemangsvillkoren i ett signerat XML-dokument. Om någon försöker att ändra abonnemangsdokumentet kommer valideringen att misslyckas.

Du kan även använda XML-signaturer för att validera nedladdade resurser som används av programmet genom att inkludera ett signerat manifest med en sammanfattning av de resurserna. Programmet kan verifiera att resurserna inte har ändrats genom att jämföra sammanfattningen i den signerade filen med en sammanfattning av de byte som hämtats. Detta är särskilt användbart när den hämtade resursen är en SWF-fil eller annat aktivt innehåll som du vill köra i programmets säkerhetssandlåda.

Grunderna i XML-signaturvalidering

Adobe AIR 1.5 och senare

Följande snabbstartartiklar på Adobe Developer Connection innehåller korta förklaringar och kodexempel på hur du validerar XML-signaturer:

- Skapa och validera XML-signaturer (Flex)
- Skapa och validera XML-signaturer (Flash)

I Adobe® AIR® används XMLSignatureValidator-klassen och IURIDereferencer-gränssnittet för validering av XML-signaturer. XML-syntaxen som stöds av XMLSignatureValidator-klassen är en deluppsättning av W3C:s rekommendation för XML-signatursyntax och -bearbetning. (Eftersom bara en deluppsättning av rekommendationen stöds kan inte alla giltiga signaturer valideras.) AIR tillhandahåller inget API för att skapa XML-signaturer.

Klasser för XML-signaturvalidering

Adobe AIR 1.5 och senare

API:t för XML-signaturvalidering inkluderar följande klasser:

Paket	Klasser
flash.security	• XMLSignatureValidator • IURIDereferencer (gränssnitt) Strängkonstanter för XMLSignatureValidator definieras i följande klasser: • ReferencesValidationSetting • RevocationCheckSettings • SignatureStatus • SignerTrustSettings
flash.events	• Event • ErrorEvent

Använda klasserna för XML-signaturvalidering

Adobe AIR 1.5 och senare

Om du vill validera en XML-signatur med hjälp av XMLSignatureValidator-klassen måste du:

- Skapa ett XMLSignatureValidator-objekt
- Ange en implementering av IURIDereferencer-gränssnittet. XMLSignatureValidator-objektet anropar `dereference()`-metoden för IURIDereferencer och skickar URI:n för varje referens i en signatur. `dereference()`-metoden måste matcha URI:n och returnera refererade data (som kan finnas i samma dokument som signaturen, eller i en extern resurs).
- Ange inställningarna för certifikatförtroende, återkallningskontroll och referensvalidering för XMLSignatureValidator-objektet så som det passar ditt program.
- Lägg till händelseavlyssnare för händelserna `complete` och `error`.
- Anropa `verify()`-metoden och ange den signatur som ska valideras.
- Hantera händelserna `complete` och `error` och tolka resultatet.

Med följande exempel implementeras en `validate()`-funktion som verifierar en XML-signaturs giltighet. XMLSignatureValidator-egenskaperna definieras så att signeringscertifikatet måste finnas i systemets förtroendearkiv eller kopplas via en kedja till ett certifikat i förtroendearkivet. Exemplet förutsätter också att det finns en lämplig IURIDereferencer-klass med namnet *XMLDereferencer*.

```
private function validate( xmlSignature:XML ):void
{
    var verifier:XMLSignatureValidator = new XMLSignatureValidator();
    verifier.addEventListener(Event.COMPLETE, verificationComplete);
    verifier.addEventListener(ErrorEvent.ERROR, verificationError);
    try
    {
        verifier.uriDereferencer = new XMLDereferencer();

        verifier.referencesValidationSetting =
            ReferencesValidationSetting.VALID_IDENTITY;
        verifier.revocationCheckSetting = RevocationCheckSettings.BEST_EFFORT;
        verifier.useSystemTrustStore = true;

        //Verify the signature
        verifier.verify( xmlSignature );
    }
    catch (e:Error)
        {
            trace("Verification error.\n" + e);
        }
}

//Trace verification results
private function verificationComplete(event:Event):void

    var signature:XMLSignatureValidator = event.target as XMLSignatureValidator;
    trace("Signature status: " + signature.validityStatus + "\n");
    trace("  Digest status: " + signature.digestStatus + "\n");
    trace("  Identity status: " + signature.identityStatus + "\n");
    trace("  Reference status: " + signature.referencesStatus + "\n");
}

private function verificationError(event:ErrorEvent):void
{
    trace("Verification error.\n" + event.text);
}
```

XML-signaturvalideringsprocessen

Adobe AIR 1.5 och senare

När du anropar `verify()`-metoden i XMLSignatureValidator, utför AIR följande steg:

- Runtime-modulen verifierar signaturens kryptografiska integritet med signeringscertifikatets offentliga nyckel.
- Runtime-modulen fastställer certifikatets kryptografiska integritet, identitet och förtroendenivå baserat på XMLSignatureValidator-objektets aktuella inställningar.

  Det förtroende som signeringscertifikatet tilldelats är avgörande för valideringsprocessens integritet. Signaturvalideringen utförs med en väldefinierad kryptografisk process, men förtroendet för signeringscertifikatet är en bedömning som inte kan göras på algoritmisk väg.

  Vanligtvis kan du avgöra om ett certifikat är tillförlitligt på tre olika sätt:

  - Genom att förlita dig på certifikatutfärdare och operativsystemets förtroendearkiv.

- Genom att, direkt från signeraren, få tillgång till en kopia av certifikatet, ett annat certifikat som fungerar som en förtroendebas för certifikatet eller tillräckligt med information för att på ett säkert sätt kunna identifiera certifikatet, t.ex. den offentliga nyckeln.
- Genom att fråga programmets slutanvändare om de litar på certifikatet. Den här typen av fråga är inte giltig med självsignerade certifikat eftersom ID-informationen i certifikatet alltid betraktas som otillförlitlig.

- Runtime-modulen verifierar den signerade informationens kryptografiska integritet.

  Den signerade informationen verifieras med hjälp av IURIDereferencer-implementeringen. För varje referens i signaturdokumentet anropas `dereference()`-metoden för IURIDereferencer-implementering. Data som returneras av `dereference()`-metoden används för att beräkna referenssammanfattningen. Det här sammanfattningsvärdet jämförs med sammanfattningen i signaturdokumentet. Om sammanfattningarna matchar varandra betyder det att informationen inte har modifierats sedan den signerades.

  Observera att endast det som signeras är säkert. Ett exempel är ett signerat manifest som specificerar filerna i ett paket. När XMLSignatureValidator verifierar signaturen kontrollerar objektet bara att själva manifestet inte har modifierats. Eftersom själva informationen i filerna inte är signerad kommer signaturen att valideras även om manifestet innehåller referenser till filer som har ändrats eller tagits bort.

  Obs! *Om du vill verifiera filer i den här typen av manifest kan du beräkna sammanfattningen för filinformationen (med samma hash-algoritm som använts i manifestet) och jämföra resultatet med sammanfattningen i det signerade manifestet. I vissa fall bör du också kontrollera om det finns ytterligare filer.*

Tolka valideringsresultatet

Adobe AIR 1.5 och senare

Valideringsresultatet rapporteras av XMLSignatureValidator-objektets statusegenskaper. Dessa egenskaper kan läsas när validator-objektet har skickat *complete*-händelsen. De fyra statusegenskaperna är: `validityStatus`, `digestStatus`, `identityStatus` och `referencesStatus`.

Egenskapen validityStatus

Adobe AIR 1.5 och senare

Egenskapen `validityStatus` visar signaturens allmänna giltighet. `validityStatus` beror på de övriga tre egenskapernas status och kan ha något av följande värden:

- `valid` – Om `digestStatus`, `identityStatus` och `referencesStatus` är `valid`.
- `invalid` – Om en eller flera av de enskilda statusegenskaperna är `invalid`.
- `unknown` – Om en eller flera av de enskilda statusegenskaperna är `unknown` och ingen enskild status är `invalid`.

Egenskapen digestStatus

Adobe AIR 1.5 och senare

Egenskapen `digestStatus` visar resultatet från den kryptografiska verifieringen av meddelandesammanfattningen. Egenskapen `digestStatus` kan ha något av följande värden:

- `valid` – Om själva signaturdokumentet inte har ändrats sedan signeringen.
- `invalid` – Om det signaturdokumentet har ändrats eller har fel format.
- `unknown` – Om metoden `verify()` inte har slutförts utan fel.

Egenskapen identityStatus

Adobe AIR 1.5 och senare

Egenskapen `identityStatus` visar signeringscertifikatets status. Den här egenskapens värde beror på flera faktorer, t.ex.:

- certifikatets kryptografiska integritet
- om certifikatet har gått ut eller återkallats
- om certifikatet betraktas som tillförlitligt på den aktuella datorn.
- XMLSignatureValidator-objektets status (t.ex. om ytterligare certifikat har lagts till för att skapa förtroendekedjan, om dessa certifikat betraktas som tillförlitliga och värdena för egenskaperna `useSystemTrustStore` och `revocationCheckSettings`)

Egenskapen `identityStatus` kan ha följande värden:

- `valid` – Signeringscertifikatet måste uppfylla följande villkor för att betraktas som giltigt:
  - Signeringscertifikatet måste vara omodifierat.
  - Signaturcertifikatet får inte ha förfallit eller återkallats, med undantag för när en giltig tidsstämpel finns i signaturen. Om signaturen är tidsstämplad anses certifikatet vara giltigt om det var giltigt vid tidpunkten då dokumentet signerades. (Certifikatet som används av tidsstämpeltjänsten för att signera tidsstämpeln måste vara kopplat till ett betrott rotcertifikat på användarens dator.)
  - Signeringscertifikatet betraktas som tillförlitligt. Ett certifikat betraktas som tillförlitligt om certifikatet finns i datorns förtroendearkiv eller om det är kopplat via en kedja till ett annat certifikat i förtroendearkivet på datorn och egenskapen `useSystemTrustStore` har värdet true. Du kan också ange att ett certifikat är tillförlitligt med hjälp av XMLSignatureValidator-objektets `addCertificate()`-metod.
  - Certifikatet är i själva verket signeringscertifikatet.
- `invalid` – Certifikatet har gått ut eller har återkallats (och det finns ingen tidsstämpel som bevisar certifikatets giltighet vid tidpunkten för signeringen) eller så har det modifierats.
- `unknown` – Om certifikatet inte är ogiltigt, men inte heller betraktas som tillförlitligt. Självsignerade certifikat visas exempelvis som `unknown` (om de inte uttryckligen har angetts som tillförlitliga). `identityStatus` visas också som `unknown` om `verify()`-metoden inte har slutförts utan fel eller om identiteten inte har kontrollerats på grund av att signatursammanfattningen är ogiltig.

Egenskapen referencesStatus

Adobe AIR 1.5 och senare

Egenskapen `referencesStatus` visar den kryptografiska integriteten för referenserna i signaturens SignedData-element.

- `valid` – Om den beräknade sammanfattningen för varje referens i signaturen matchar motsvarande sammanfattning i XML-signaturen. Statusen `valid` anger att den signerade informationen inte har modifierats.
- `invalid` – Om en beräknad sammanfattning inte matchar motsvarande sammanfattning i signaturen.
- `unknown` – Om referenssammanfattningarna inte har kontrollerats. Referenserna kontrolleras inte om den övergripande signatursammanfattningen är `invalid` eller om signeringscertifikatet är ogiltigt. Om `identityStatus` har värdet `unknown` kontrolleras bara referenserna om `referencesValidationSetting` har värdet `validOrUnknown`.

Om XML-signaturer

Adobe AIR 1.5 och senare

En XML-signatur är en digital signatur som representeras i XML-syntaxen. Informationen i en XML-signatur kan användas för att verifiera att den signerade informationen inte har modifierats sedan signeringen. När ett signeringscertifikat har utfärdats av en betrodd certifikatutfärdare kan signerarens identitet dessutom verifieras via infrastrukturen med offentliga nycklar.

En XML-signatur kan användas med alla typer av digitala data (i binärt format eller XML-format). XML-signaturer används exempelvis för att:

- kontrollera om externa eller hämtade resurser har ändrats
- verifiera att meddelanden kommer från en känd källa
- validera programlicens- eller prenumerationsbehörigheter.

XML-signatursyntax som stöds

Adobe AIR 1.5 och senare

AIR stöder följande element från W3C:s rekommendation för XML-signatursyntax och -bearbetning:

- Alla grundläggande signatursyntaxelement (avsnitt 4 i W3C:s rekommendationsdokument), förutom elementet KeyInfo som inte stöds fullt ut
- Elementet KeyInfo får bara innehålla ett X509Data-element
- Ett X509Data-element får bara innehålla ett X509Certificate-element
- SHA256-sammanfattningsmetoden
- RSA-SHA1-algoritmen (PKCS1) för signering
- Transformeringsalgoritmen och kanoniseringsmetoden ”Kanonisk XML utan kommentarer”
- Transformeringen av inbäddade (enveloped) signaturer
- tidstämplar

Följande dokument illustrerar en typisk XML-signatur (större delen av den kryptografiska informationen har tagits bort för att förenkla exemplet):

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="URI_to_signed_data">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/></Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>uoo...vY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>Ked...w==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>i7d...w==</X509Certificate>
        </X509Data>
    </KeyInfo>
</Signature>
```

Nyckelelementen i en signatur är:

- SignedInfo – Innehåller referenser till den signerade informationen och de beräknade sammanfattningsvärdena vid tidpunkten för signeringen. Den signerade informationen kan vara extern eller finnas i samma dokument som XML-signaturen.
- SignatureValue – Innehåller en sammanfattning av SignedInfo-elementet som krypterats med signerarens privata nyckel.
- KeyInfo – Innehåller signeringscertifikatet och eventuella andra certifikat som behövs för att upprätta förtroendekedjan. Lägg märke till att även om KeyInfo-elementet tekniskt sett är valfritt kan AIR inte validera signaturen om den inte är inkluderad.

Det finns tre allmänna typer av XML-signaturer:

- Enveloped – signaturen infogas i den XML-information som den signerar.
- Enveloping – den signerade XML-informationen finns i ett Object-element i Signature-elementet.
- Detached – den signerade informationen är extern och finns inte i samma dokument som XML-signaturen. Den signerade informationen kan finnas i en extern fil. Informationen kan också finnas i samma XML-dokument som signaturen, i stället för som ett överordnat eller underordnat element till Signature-elementet.

XML-signaturer använder URI:er för att referera till den signerade informationen. Signerings- och valideringsprogrammen måste använda samma konventioner för matchningen av dessa URI:er. När du använder XMLSignatureValidator-klassen måste du ange en implementering av IURIDereferencer-gränssnittet. Den här implementeringen ansvarar för att matcha URI:n och returnerar den signerade informationen som ett ByteArray-objekt. Det returnerade ByteArray-objektet sammanfattas med samma algoritm som användes för att generera sammanfattningen i signaturen.

Certifikat och förtroende

Adobe AIR 1.5 och senare

Ett certifikat består av en offentlig nyckel, ID-information och sannolikt ett eller flera certifikat som tillhör certifikatutfärdaren.

Du kan skapa förtroende i ett certifikat på två sätt. Du kan upprätta förtroende genom att erhålla en kopia av certifikatet från signeraren, t.ex. på ett fysiskt medium, eller via en säker digital överföring, t.ex. en SSL-överföring. Du kan också förlita dig på en certifikatutfärdare för att avgöra om signeringscertifikatet är tillförlitligt.

I det senare fallet måste signeringscertifikatet vara utfärdat av en utfärdare som är betrodd på den dator som signaturen valideras på. De flesta tillverkare av operativsystem placerar rotcertifikaten för ett antal certifikatutfärdare i operativsystemets förtroendearkiv. Användaren kan lägga till eller ta bort certifikat i arkivet.

Även om ett certifikat har utfärdats av en betrodd certifikatutfärdare måste du fortfarande bestämma om certifikatet tillhör någon som du litar på. I många fall fattas detta beslut av slutanvändaren. När ett AIR-program installeras visas exempelvis ID-informationen från utfärdarens certifikat när användaren ombeds att bekräfta att han eller hon vill installera programmet. I andra fall kan du behöva jämföra den offentliga nyckeln eller annan certifikatinformation med en lista med godkända nycklar. (Den här listan måste vara skyddad, t.ex. med sin egen signatur eller genom att den lagras på den krypterade lokala lagringsplatsen i AIR, så att det inte går att manipulera själva listan.)

Obs! *Du kan välja att lita på signeringscertifikatet utan separat verifiering, t.ex. om en signatur är självsignerad, men verifieringen av signaturen ger i detta fall ingen garanti avseende säkerheten. Om du inte vet vem som skapade signaturen är det svårt, om inte omöjligt, att försäkra att signaturen inte har manipulerats. Signaturen kan vara en signerad förfalskning.*

Återkallning och utgångsdatum för certifikat

Adobe AIR 1.5 och senare

Alla certifikat går ut. Certifikat kan också återkallas av certifikatutfärdaren, t.ex. om certifikatets privata nyckel har kompromettterats eller blivit stulen. Om en signatur har signerats med ett certifikat som gått ut eller som har återkallats visas signaturen som ogiltig, såvida signaturen inte innehåller en tidsstämpel. Om signaturen innehåller en tidsstämpel valideras signaturen av XMLSignatureValidator-klassen, förutsatt att certifikatet var giltigt vid tidpunkten för signeringen.

En tidsstämpel är ett signerat digitalt meddelande från en tidsstämpeltjänst som certifierar att informationen signerades vid en viss tid ett visst datum. Tidsstämplar utfärdas av tidsstämpelutfärdare och signeras med tidsstämpelutfärdarens eget certifikat. Tidsstämpelutfärdarens certifikat som är inbäddat i tidsstämpeln måste betraktas som tillförlitligt på den aktuella datorn för att tidsstämpeln ska betraktas som giltig. XMLSignatureValidator tillhandahåller inget API för att ange ett annat certifikat som kan användas vid valideringen av tidsstämpeln.

Implementera IURIDereferencer-gränssnittet

Adobe AIR 1.5 och senare

För att validera en XML-signatur måste du ange en implementering av IURIDereferencer-gränssnittet. Implementeringen matchar URI:erna i Reference-elementen i ett XML-signaturdokument och returnerar informationen så att sammanfattningen kan beräknas. Den beräknade sammanfattningen jämförs med sammanfattningen i signaturen för att avgöra om den refererade informationen har modifierats sedan signaturen skapades.

Obs! *HTML-baserade AIR-program måste importera ett SWF-bibliotek som innehåller en ActionScript-implementering för att XML-signaturer ska kunna valideras. IURIDereferencer-gränssnittet kan inte implementeras i JavaScript.*

IURIDerefencer-gränssnittet har en enda metod, `dereference(uri:String)`, som måste implementeras. XMLSignatureValidator-objektet anropar den här metoden för varje referens i signaturen. Metoden måste returnera informationen i ett ByteArray-objekt.

I de flesta fall måste du även lägga till egenskaper eller metoder som gör att IURIDereferencer-objektet kan hitta den refererade informationen. Om den signerade informationen exempelvis finns i samma dokument som signaturen kan du lägga till en medlemsvariabel som innehåller en referens till XML-dokumentet. Metoden `dereference()` kan sedan använda den här variabeln, jämte URI:n, för att hitta informationen som refereras. På samma sätt, om den signerade informationen finns i en katalog i det lokala filsystemet, kan `dereference()`-metoden behöva en egenskap som innehåller sökvägen till katalogen för att kunna matcha de refererade filerna.

XMLSignatureValidator förlitar sig helt och hållet på IURIDereferencer-objektet för att tolka URI-strängar. Standardreglerna för dereferencing-URI:er anges i avsnitt 4.3.3 i W3C:s rekommendation för XML-signatursyntax och -bearbetning.

IURIDereferencer-URI:er i inbäddade (enveloped) signaturer

Adobe AIR 1.5 och senare

När en inbäddad XML-signatur genereras, infogas signaturelementen i den signerade informationen. Om du exempelvis har signerat följande meddelande med en struktur för en inbäddad signatur:

```
<message>
    <data>...</data>
</message>
```

Ser det resulterande signerade dokumentet ut så här:

```
<message>
    <data>...</data>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>yv6...Z0Y=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>cCY...LQ==</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509Certificate>MII...4e</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</message>
```

Observera att signaturen innehåller ett enda Reference-element med en tom sträng som dess URI. En tom sträng i den här kontexten refererar till dokumentets rot.

Observera också att transformeringsalgoritmen anger att en transformering av en inbäddad signatur har tillämpats. När den här typen av transformering har tillämpats tar XMLSignatureValidator automatiskt bort signaturen i dokumentet innan sammanfattningen beräknas. Det betyder att IURIDereferencer inte behöver ta bort Signature-elementet när informationen returneras.

Följande exempel illustrerar en IURIDereferencer för inbäddade signaturer:

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class EnvelopedDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var signedMessage:XML;

        public function EnvelopedDereferencer( signedMessage:XML )
        {
            this.signedMessage = signedMessage;
        }

        public function dereference( uri:String ):IDataInput
        {
            try
            {
                if( uri.length != 0 )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var data:ByteArray = new ByteArray();
                data.writeUTFBytes( signedMessage.toXMLString() );
                data.position = 0;
            }
            catch (e:Error)
                {
                var error:ErrorEvent =
                    new ErrorEvent("Ref error " + uri + " ", false, false, e.message);
                this.dispatchEvent(error);
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }
        }
    }
}
```

Den här IURIDereferencer-klassen använder en konstruktorfunktion med en parameter, `signedMessage`, för att göra dokumentet med den inbäddade signaturen tillgängligt för `dereference()`-metoden. Eftersom referensen i en inbäddad signatur alltid refererar till roten för den signerade informationen skriver `dereferencer()`-metoden dokumentet till en bytearray och returnerar ByteArray-objektet.

IURIDereferencer-URI:er i inbäddade (enveloped) och separata (detached) signaturer

Adobe AIR 1.5 och senare

Om den signerade informationen finns i samma dokument som själva signaturen använder URI:erna i referenserna vanligtvis XPath- eller XPointer-syntax för att hantera elementen som signerats. W3C:s rekommendation för XML-signatursyntax och -bearbetning rekommenderar bara den här syntaxen. Det betyder att du bör basera implementeringen på de signaturer som du förväntar dig att hitta (och lägga till nödvändiga felkontroller för att hantera syntax som inte stöds).

Signaturen för ett AIR-program är ett exempel på en inbäddad signatur. Filerna i programmet visas i ett Manifest-element. Manifest-elementet hanteras i Reference URI-attributet med strängen "#PackageContents", som refererar till Manifest-elementets ID:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="PackageSignature">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/TR/xmldsig-core#rsa-sha1"/>
        <Reference URI="#PackageContents">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>ZMGqQdaRKQc1HirIRsDpeBDlaElS+pPotdziIAyAYDk=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue Id="PackageSignatureValue">cQK...7Zg==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>MII...T4e</X509Certificate>
        </X509Data>
    </KeyInfo>
    <Object>
    <Manifest Id="PackageContents">
        <Reference URI="mimetype">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>0/oCb84THKMagtI0Dy0KogEu92TegdesqRr/clXct1c=</DigestValue>
        </Reference>
        <Reference URI="META-INF/AIR/application.xml">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>P9MqtqSdqcqnFgeoHCJysLQu4PmbUW2JdAnc1WLq8h4=</DigestValue>
        </Reference>
        <Reference URI="XMLSignatureValidation.swf">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>OliRHRAgc9qt3Dk0m0Bi53Ur5ur3fAweIFwju74rFgE=</DigestValue>
        </Reference>
    </Manifest>
</Object>
</Signature>
```

Ett IURIDereferencer-objekt som används för att validera den här signaturen måste hämta URI-strängen som innehåller `"#PackageContents"` från Reference-elementet och returnera Manifest-elementet i ett ByteArray-objekt. Symbolen "#" refererar till attributvärdet för ett element-ID.

Följande exempel implementerar ett IURIDereferencer-objekt för validering av AIR-programsignaturer. Implementeringen i exemplet baseras på den kända strukturen för en AIR-signatur. En mer allmängiltig IURIDereferencer kan vara mycket mer komplex.

```
package
{
    import flash.events.ErrorEvent;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class AIRSignatureDereferencer implements IURIDereferencer {
        private const XML_SIG_NS:Namespace =
            new Namespace( "http://www.w3.org/2000/09/xmldsig#" );
        private var airSignature:XML;

        public function AIRSignatureDereferencer( airSignature:XML ) {
            this.airSignature = airSignature;
        }

        public function dereference( uri:String ):IDataInput {
            var data:ByteArray = null;
            try
            {
                if( uri != "#PackageContents" )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var manifest:XMLList =
                    airSignature.XML_SIG_NS::Object.XML_SIG_NS::Manifest;
                data = new ByteArray();
                data.writeUTFBytes( manifest.toXMLString());
                data.position = 0;
            }
            catch (e:Error)
            {
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }

        }
    }
}
```

När du verifierar den här typen av signatur valideras bara informationen i Manifest-elementet. Själva filerna i paketet kontrolleras inte över huvud taget. Om du vill kontrollera om filerna i paketet har manipulerats måste du läsa filerna, beräkna SHA256-sammanfattningen och jämföra resultatet med sammanfattningen i manifestet. XMLSignatureValidator kontrollerar inte automatiskt den här typen av sekundära referenser.

Obs! *Det här exemplet är endast avsett att illustrera signaturvalideringsprocessen. Ett AIR-program som validerar sin egen signatur är inte särskilt användbart. Om programmet redan har manipulerats skulle valideringskontrollen enkelt kunna tas bort.*

Beräkna sammanfattningsvärden för externa resurser

Adobe AIR 1.5 och senare

AIR innehåller inga inbyggda funktioner för beräkning av SHA256-sammanfattningar, men Flex SDK inkluderar en SHA256-verktygsklass. SDK-paketet inkluderar också Base64-kodverktygsklassen som är praktisk för att jämföra den beräknade sammanfattningen med sammanfattningen i en signatur.

Följande exempelfunktion läser och validerar filerna i ett AIR-paketmanifest:

```
import mx.utils.Base64Encoder;
import mx.utils.SHA256;

private function verifyManifest( sigFile:File, manifest:XML ):Boolean
{
    var result:Boolean = true;
    var message:String = '';
    var nameSpace:Namespace = manifest.namespace();

    if( manifest.nameSpace::Reference.length() <= 0 )
    {
        result = false;
        message = "Nothing to validate.";
    }
    for each (var reference:XML in manifest.nameSpace::Reference)
    {
        var file:File = sigFile.parent.parent.resolvePath( reference.@URI );
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var fileData:ByteArray = new ByteArray();
        stream.readBytes( fileData, 0, stream.bytesAvailable );

        var digestHex:String = SHA256.computeDigest( fileData );
        //Convert hexidecimal string to byte array
        var digest:ByteArray = new ByteArray();
        for( var c:int = 0; c < digestHex.length; c += 2 ){
            var byteChar:String = digestHex.charAt(c) + digestHex.charAt(c+1);
            digest.writeByte( parseInt( byteChar, 16 ));
        }
        digest.position = 0;
```

```
        var base64Encoder:Base64Encoder = new Base64Encoder();
        base64Encoder.insertNewLines = false;
        base64Encoder.encodeBytes( digest, 0, digest.bytesAvailable );
        var digestBase64:String = base64Encoder.toString();
        if( digestBase64 == reference.nameSpace::DigestValue )
        {
            result = result && true;
            message += "   " + reference.@URI + " verified.\n";
        }
        else
        {
            result = false;
            message += " ---- " + reference.@URI + " has been modified!\n";
        }
        base64Encoder.reset();
    }
    trace( message );
    return result;
}
```

Funktionen loopar genom alla referenserna i Manifest-elementet. För varje referens beräknas SHA256-sammanfattningen, kodad i base64-format, och jämförs med sammanfattningen i manifestet. URI:erna i ett AIR-paket refererar till sökvägar som är relativa till programkatalogen. Sökvägarna matchas baserat på platsen för signaturfilen, som alltid finns i META-INF-underkatalogen i programkatalogen. Observera att Flex SHA256-klassen returnerar sammanfattningen som en sträng med hexadecimala tecken. Den här strängen måste konverteras till en bytearray som innehåller de byte som representeras i den hexadecimala strängen.

Om du vill använda mx.utils.SHA256- och Base64Encoder-klasserna i Flash CS4 kan du antingen leta upp och kopiera klasserna i programutvecklingskatalogen eller kompilera en biblioteks-SWF-fil som innehåller klasserna med hjälp av Flex SDK.

IURIDereferencer-URI:er i separata (detached) signaturer med referenser till externa data

Adobe AIR 1.5 och senare

Om en URI refererar till en extern resurs måste informationen hämtas och läsas in i ett ByteArray-objekt. Om URI:n innehåller en absolut URL behöver du bara läsa en fil eller begära en URL. Om, vilket förmodligen är vanligare, URI:n innehåller referenser till en relativ sökväg måste IURIDereferencer-implementeringen specificera hur sökvägarna ska matchas i de signerade filerna.

I följande exempel används ett File-objekt som initieras när IURIDereferencer-instansen konstrueras som utgångspunkt för matchningen av signerade filer.

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.filesystem.File;
    import flash.filesystem.FileMode;
    import flash.filesystem.FileStream;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;
    public class RelativeFileDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var base:File;

        public function RelativeFileDereferencer( base:File )
        {
            this.base = base;
        }

        public function dereference( uri:String ):IDataInput
        {
            var data:ByteArray = null;
            try{
                var referent:File = this.base.resolvePath( uri );
                var refStream:FileStream = new FileStream();
                data = new ByteArray();
                refStream.open( referent, FileMode.READ );

                refStream.readBytes( data, 0, data.bytesAvailable );

            } catch ( e:Error ) {
                data = null;
                throw new Error("Reference not resolvable: " + referent.nativePath + ", " +
e.message );
            } finally {
                return data;
            }
        }
    }
}
```

Funktionen `dereference()` letar upp filen som anges av referens-URI:n, läser in filinnehållet i en bytearray och returnerar ByteArray-objektet.

Obs! *Innan du validerar externa fjärreferenser bör du överväga om ditt program är sårbart för en "phone home" eller andra typer av angrepp av ett signaturdokument som skapats i syfte att skada.*

Kapitel 49: Klientsystemmiljö

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I det här avsnittet beskrivs hur du interagerar med användarens dator. Kapitlet innehåller information om hur du avgör vilka funktioner som stöds samt hur du skapar flerspråkiga program med hjälp av användarens installerade redigeringsprogram för inmatningsmetod (IME-program) om ett sådant finns. Kapitlet innehåller även beskrivningar av typiska situationer när programdomäner kan användas.

Fler hjälpavsnitt

flash.system.System

flash.system.Capabilities

Grundläggande om klientsystemmiljön

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du skapar mer avancerade program kanske du behöver känna till mer om användarnas operativsystem och behöver få tillgång till funktioner i deras operativsystem. Paketet flash.system innehåller en samling klasser som gör att du får åtkomst till funktioner på systemnivån som exempelvis följande:

- Avgöra vilken program- och säkerhetsdomän som koden körs i
- Avgöra egenskaperna hos användarens instans för Flash-körning (till exempel Flash® Player eller Adobe® AIR™), till exempel skärmstorlek (upplösning) och om vissa funktioner är tillgängliga, till exempel MP3-ljud
- Skapa flerspråkiga platser med hjälp av IME-program
- Interagera med Flash-körningens behållare (till exempel en HTML-sida eller ett behållarprogram).
- Spara information till användarens Urklipp

Paketet flash.system innehåller även klasserna IMEConversionMode och SecurityPanel. Klasserna innehåller statiska konstanter som du använder med klassen IME respektive klassen Security.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer:

Operativsystem Huvudprogrammet som körs på en dator och i vilket alla övriga program körs. Exempel på operativsystem är Microsoft Windows, Mac OS X och Linux®.

Urklipp Operativsystemets behållare för att lagra text eller objekt som har kopierats eller klippts ut och från vilken objekt sedan kan klistras in i program.

Programdomän En mekanism för att dela in klasser som används i olika SWF-filer. Därmed går det att undvika att klasser skriver över varandra när SWF-filer innehåller olika klasser med samma namn.

IME (redigeringsprogram för inmatningsmetod) Ett program (eller operativsystemverktyg) som används för att ange specialtecken eller symboler med hjälp av ett vanligt tangentbord.

Klientsystem Inom programmering är klienten en del av ett program (eller ett helt program) som körs på en användarens dator och som används av en enskild användare. Klientsystemet är det underliggande operativsystemet på användarens dator.

Använda klassen System

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen System innehåller metoder och egenskaper som gör att du kan interagera med användarens operativsystem och inhämta information om aktuell minnesanvändning för körningen. Metoderna och egenskaperna för klassen System gör att du även kan lyssna efter `imeComposition`-händelser, instruera att externa textfiler med användarens aktuella teckentabell ska läsas in under körningen eller läsas in som Unicode och ställa in innehållet i användarens urklipp.

Hämta data om användarens system under körning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Genom att kontrollera egenskapen `System.totalMemory` kan du se hur mycket minne (i byte) som används för närvarande under körningen. Egenskapen gör att du kan övervaka minnesanvändningen och optimera dina program utifrån hur minnesnivån förändras. Om till exempel en visuell effekt kräver mycket minne, kanske du behöver ändra effekten eller ta bort den helt och hållet.

Egenskapen `System.ime` är en referens till det IME-program som är installerat. Egenskapen gör att du kan lyssna efter `imeComposition`-händelser (`flash.events.IMEEvent.IME_COMPOSITION`) med hjälp av metoden `addEventListener()`.

Den tredje egenskapen i klassen System är `useCodePage`. När `useCodePage` är inställt på `true` används under körningen operativsystemets traditionella teckentabell vid inläsning av externa textfiler. Om du ställer in den här egenskapen på `false` kommer under körningen den externa filen att tolkas som Unicode.

Kom ihåg att om du ställer in `System.useCodePage` på `true` måste den traditionella teckentabellen i operativsystemet innehålla de tecken som används i den externa textfilen för att texten ska kunna visas. Om du till exempel läser in en extern textfil som innehåller kinesiska tecken, kan dessa tecken inte visas i ett system som har teckentabellen för engelskt Windows eftersom den teckentabellen inte innefattar kinesiska tecken.

För att användare på alla plattformar ska kunna se externa textfiler som används i dina program, måste du koda alla externa textfiler som Unicode och låta `System.useCodePage` som standard vara inställt på `false`. På det här sättet tolkas texten som Unicode under körningen.

Spara text till urklipp

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen System innehåller en metod med namnet `setClipboard()` som gör att Flash-körningen kan ställa in innehållet i användarens urklipp med en angiven sträng. Av säkerhetsskäl finns det ingen `Security.getClipboard()`-metod, eftersom en sådan metod skulle kunna göra att oseriösa webbplatser kan få tillgång till de data som senast har kopierats till användarens urklipp.

Följande kod visar hur ett felmeddelande kan kopieras till användarens urklipp när ett säkerhetsfel uppstår. Felmeddelandet kan vara användbart om användaren vill rapportera ett möjligt programfel för ett program.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Flash Player 10 och AIR 1.0

Du kan använda klassen Clipboard om du vill läsa och skriva urklippsdata som svar på en användarhändelse. I AIR krävs ingen användarhändelse för kod som körs i programsandlådan för att få åtkomst till urklipp.

Använda klassen Capabilities

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klassen Capabilities gör att utvecklare kan fastställa i vilken miljö ett program körs. Genom att använda olika egenskaper i klassen Capabilities kan du ta reda på användarsystemets upplösning, om användarens system stöder programvara för hjälpmedel, språkversionen av användarens operativsystem och vilken version av Flash-körningen som är installerad.

Genom att kontrollera egenskaperna i klassen Capabilities kan du anpassa ditt program så att fungerar så bra som möjligt med användarens miljö. Om du till exempel kontrollerar egenskaperna `Capabilities.screenResolutionX` och `Capabilities.screenResolutionY`, kan du fastställa skärmupplösningen i användarens system och bestämma vilken videostorleken som är lämpligast. Du kan även kontrollera egenskapen `Capabilities.hasMP3` för att undersöka om användarens system stöder MP3-uppspelning innan ett försök görs att läsa in en extern MP3-fil.

Följande kod använder ett reguljärt uttryck för att analysera Flash-körningsversionen som klienten använder:

```
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.");
}
```

Om du vill skicka användarens systemegenskaper till ett skript på servern så att informationen kan lagras i en databas, kan du använda följande ActionScript-kod:

```
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);
```

Exempel på egenskaper: identifera systemegenskaper

Flash Player 9 och senare

Exemplet CapabilitiesExplorer visar hur du kan använda klassen flash.system.Capabilities för att kontrollera vilka funktioner som stöds i användarens version av Flash-körningen. I exemplet visas följande tekniker:

- Identifiera vilka egenskaper användarens version av Flash-körningen stöder genom att använda klassen Capabilities
- Använda klassen ExternalInterface för att känna av vilka webbläsarinställningar användarens webbläsare stöder

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. CapabilitiesExplorer-programfilerna finns i mappen Samples/CapabilitiesExplorer. Programmet består av följande filer:

Fil	Beskrivning
CapabilitiesExplorer.fla eller CapabilitiesExplorer.mxml	Huvudprogramfilen i Flash (FLA) eller Flex (MXML).
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	Klassen som tillhandahåller programmets huvudfunktioner, inklusive att lägga till systemegenskaperna till en array, sortera objekten och använda klassen ExternalInterface för att hämta webbläsaregenskaper.
capabilities.html	En HTML-behållare som innehåller nödvändigt JavaScript för att kommunicera med det externa API:t.

Översikt över CapabilitiesExplorer

Flash Player 9 och senare

Filen CapabilitiesExplorer.mxml ansvarar för att ställa in användargränssnittet för programmet CapabilitiesExplorer. Funktionerna i användarens version av Flash-körningen visas i en instans av en DataGrid-komponent på scenen. Användarens webbläsaregenskaper visas också om användaren kör programmet från en HTML-behållare och om det externa API:t är tillgängligt.

När huvudprogramfilens händelse `creationComplete` skickas, anropas metoden `initApp()`. Metoden `initApp()` anropar metoden `getCapabilities()` från klassen com.example.programmingas3.capabilities.CapabilitiesGrabber. Koden för metoden `initApp()` är följande:

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

Metoden `CapabilitiesGrabber.getCapabilities()` returnerar en sorterad array med Flash-körningens och webbläsarens egenskaper, som sedan ställs in på egenskapen `dataProvider` för DataGrid-komponentinstansen `capabilitiesGrid` på scenen.

Översikt över klassen CapabilitiesGrabber

Flash Player 9 och senare

Den statiska metoden `getCapabilities()` i klassen CapabilitiesGrabber lägger till alla egenskaper från klassen flash.system.Capabilities till en array (`capDP`). Sedan anropar den statiska metoden `getBrowserObjects()` i klassen CapabilitiesGrabber. Metoden `getBrowserObjects()` använder det externa API:t för att göra en slinga över webbläsarens navigerarobjekt, som innehåller webbläsarens egenskaper. Koden för metoden `getCapabilities()` är följande:

```
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;
}
```

Metoden `getBrowserObjects()` returnerar en array med alla egenskaper i webbläsarens navigerarobjekt. Om arrayen omfattar minst ett objekt, bifogas arrayen med webbläsaregenskaper (`navArr`) till arrayen med Flash Player-egenskaper (`capDP`) och sedan sorteras hela arrayen i alfabetisk ordning. Slutligen returneras den sorterade arrayen till huvudprogramfilen, som fyller datarastret. Koden för metoden `getBrowserObjects()` är följande:

```
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;
}
```

Om det externa API:t är tillgängligt i den aktuella användarmiljön, anropar Flash-körningen JavaScript-metoden `JS_getBrowserObjects()`, som gör en slinga över webbläsarens navigerarobjekt och returnerar en sträng med URL-kodade värden till ActionScript. Strängen konverteras sedan till ett URLVariables-objekt (`itemVars`) och läggs in i arrayen `itemArr`, som returneras till det anropande skriptet.

Kommunicera med JavaScript

Flash Player 9 och senare

Det sista momentet när du skapar CapabilitiesExplorer-programmet går ut på att skriva den JavaScript-kod som gör en slinga över objekten i webbläsarens navigerarobjekt och bifogar ett namnvärdepar till en tillfällig array. Koden för JavaScript-metoden `JS_getBrowserObjects()` i filen container.html är följande:

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
```

```
                    // escape() function.
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
                }
            }
            // Loop over each item in the browser's screen object.
            for (var name in screen)
            {
                var value = screen[name];

                // If the current value is a number, add it to the array, otherwise
                // ignore the item.
                switch (typeof(value))
                {
                    case "number":
                        var tempStr = "screen." + name + "=" + escape(value);
                        tempArr.push(tempStr);
                        break;
                }
            }
            // Return the array as a URL-encoded string of name-value pairs.
            return tempArr.join("&");
        }
</script>
```

Koden inleds med att skapa en tillfällig array som ska innehålla alla namnvärdepar i navigerarobjektet. Sedan görs en slinga över navigerarobjektet med en `for..in`-slinga, och datatypen för det aktuella värdet utvärderas för att filtrera bort oönskade värden. I det här programmet är vi bara intresserade av String- och Boolean-värden. Övriga datatyper (som funktioner och arrayer) ignoreras. Varje String- eller Boolean-värde i navigerarobjektet bifogas till arrayen `tempArr`. Sedan görs en slinga över webbläsarobjektet med hjälp av en `for..in`-slinga och alla numeriska värden läggs till i arrayen `tempArr`. Slutligen konverteras den tillfälliga arrayen till en sträng med metoden `Array.join()`. Arrayen använder ett et-tecken (&) som avgränsare, vilket gör att ActionScript enkelt kan analysera data med hjälp av klassen URLVariables.

Kapitel 50: Anropa och avsluta AIR-program

Adobe AIR 1.0 och senare

I det här avsnittet behandlas hur ett installerat Adobe® AIR®-program kan anropas liksom alternativ och överväganden vid avslutande av ett program som körs.

Obs! *Objekten NativeApplication, InvokeEvent och BrowserInvokeEvent är bara tillgängliga för SWF-innehåll som körs i AIR-programmets sandlåda. SWF-innehåll som körs i Flash Player, antingen i webbläsaren eller i den fristående spelaren (projektorn), eller i ett AIR-program utanför programmets sandlåda, kan inte komma åt de här klasserna.*

Följande snabbstartartiklar på Adobe Developer Connection innehåller information med korta förklaringar och kodexempel på hur du anropar och avslutar AIR-program:

- Startalternativ

Fler hjälpavsnitt

flash.desktop.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

Anropa program

Adobe AIR 1.0 och senare

Ett AIR-program anropas när användaren (eller operativsystemet):

- Startar programmet från skrivbordsskalet.
- Använder programmet som ett kommando i ett kommandoradsskal.
- Öppnar en typ av fil som programmet är standardprogram för.
- (Mac OS X) Klickar på programikonen i aktivitetsfältet (oavsett om programmet körs eller inte).
- Väljer att starta programmet från installationsprogrammet (antingen i slutet av en ny installationsprocess eller efter att ha dubbelklickat på AIR-filen för ett redan installerat program).
- Börjar en uppdatering av ett AIR-program när den installerade versionen har signalerat att den hanterar programuppdateringar själv (genom att ta med en `<customUpdateUI>true</customUpdateUI>`-deklaration i programbeskrivningsfilen).
- (iOS) Erhåller ett meddelande från APN-tjänster (Apple Push Notification).
- Aktiverar programmet via en URL-adress.

- Besöker en webbsida som är värd för ett Flash-emblem eller ett program som anropar metoden `com.adobe.air.AIR launchApplication()` som anger ID-information för AIR-programmet. (Programbeskrivningen måste också omfatta en `<allowBrowserInvocation>true</allowBrowserInvocation>`-deklaration om webbläsaranrop ska lyckas.)

När ett AIR-program anropas skickas ett InvokeEvent-objekt av typen `invoke` via singleton-objektet NativeApplication. För att en programtid ska kunna initiera sig själv och registrera en händelseavlyssnare måste `invoke`-händelser ställas i kö i stället för att tas bort. Så snart en avlyssnare är registrerad levereras alla händelser som står i kö.

Obs! *När ett program anropas med hjälp av webbläsarens anropsfunktion skickar objektet NativeApplication bara en `invoke`-händelse om programmet inte redan körs.*

Om du vill ta emot `invoke`-händelser måste du anropa NativeApplication-objektets `addEventListener()`-metod (`NativeApplication.nativeApplication)`. När en händelseavlyssnare registrerar sig till en `invoke`-händelse, kommer den också att få alla `invoke`-händelser som har inträffat före registreringen. `invoke`-händelser som står i kö skickas en i taget med korta intervall efter att anropet till `addEventListener()` returnerats. Om en ny `invoke`-händelse inträffar under den här processen kan den skickas före en eller flera av händelserna som står i kö. Tack vare det här kösystemet kan du hantera `invoke`-händelser som inträffar innan initieringskoden körs. Kom ihåg, att om du lägger till en händelseavlyssnare senare i körningen (efter programinitieringen) kommer den fortfarande att få alla `invoke`-händelser som har inträffat sedan programmet startade.

Bara en instans av ett AIR-program startar. När ett program som redan körs anropas igen, skickas en ny `invoke`-händelse till den instans som körs. Det åligger ett AIR-program att svara på en `invoke`-händelse och vidta rätt åtgärd (till exempel att öppna ett nytt dokumentfönster).

Ett `InvokeEvent`-objekt innehåller alla argument som skickas till programmet samt katalogen som programmet anropas från. Om programmet anropades med anledning av av en filtypsassociation, ingår den fullständiga sökvägen till filen i kommandoradsargumenten. Om programmet anropades med anledning av en programuppdatering anges också den fullständiga sökvägen till uppdateringsfilen.

Om flera filer öppnas i en åtgärd skickas ett enskilt InvokeEvent-objekt på Mac OS X. Varje fil inkluderas i `argument`-arrayen. På Windows och Linux skickas ett separat InvokeEvent-objekt för varje fil.

Programmet kan hantera `invoke`-händelser genom att registrera en avlyssnare med NativeApplication-objektet:

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
```

Och definiera en händelseavlyssnare:

```
 var arguments:Array;
var currentDir:File;
public function onInvokeEvent(invocation:InvokeEvent):void {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

Hämta kommandoradsargument

Adobe AIR 1.0 och senare

Kommandoradsargumenten som hör till ett AIR-programanrop levereras i InvokeEvent-objektet, som skickas av objektet NativeApplication. Egenskapen `arguments` för InvokeEvent innehåller en array med de argument som skickas av operativsystemet när ett AIR-program anropas. Om argumenten innehåller relativa filsökvägar kan du vanligtvis lösa sökvägen med hjälp av egenskapen `currentDirectory`.

De argument som skickas till ett AIR-program behandlas som blankstegsavgränsade strängar, såvida de inte omsluts av citattecken:

Argument	Array
tick tock	{tick,tock}
tick "tick tock"	{tick,tick tock}
"tick" "tock"	{tick,tock}
\"tick\" \"tock\"	{"tick","tock"}

Egenskapen `currentDirectory` för ett InvokeEvent-objekt innehåller ett File-objekt, som representerar den katalog programmet startas från.

När ett program anropas med anledning av att en fil av en typ som är registrerad av programmet öppnas, ingår den ursprungliga sökvägen till filen i kommandoradsargumenten som en sträng. (Programmet ansvarar för att öppna eller utföra den tänkta filåtgärden.) När ett program är programmerat att uppdatera sig självt (i stället för att använda det vanliga uppdateringsgränssnittet för AIR), inkluderas den ursprungliga sökvägen till AIR-filen när en användare dubbelklickar på en AIR-fil som finns i ett program med motsvarande program-ID.

Du får tillgång till filen med metoden `resolve()` för File-objektet `currentDirectory`:

```
 if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

Du bör också kontrollera att ett argument verkligen är sökvägen till en fil.

Exempel: Anropa händelseloggen

Adobe AIR 1.0 och senare

I följande exempel visas hur du registrerar händelseavlyssnare för och hanterar händelsen `invoke`. Exemplet loggar alla anropshändelser som tas emot och visar den aktuella katalogen och kommandoradsargumenten.

ActionScript-exempel

```
package
{
    import flash.display.Sprite;
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;
    import flash.text.TextField;

    public class InvokeEventLogExample extends Sprite
    {
        public var log:TextField;

        public function InvokeEventLogExample()
        {
            log = new TextField();
            log.x = 15;
            log.y = 15;
            log.width = 520;
            log.height = 370;
            log.background = true;

            addChild(log);

            NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvoke);
        }

        public function onInvoke(invokeEvent:InvokeEvent):void
        {
            var now:String = new Date().toTimeString();
            logEvent("Invoke event received: " + now);

            if (invokeEvent.currentDirectory != null)
            {
                logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
            }
            else
            {
```

```
                logEvent("--no directory information available--");
            }

            if (invokeEvent.arguments.length > 0)
            {
                logEvent("Arguments: " + invokeEvent.arguments.toString());
            }
            else
            {
                logEvent("--no arguments--");
            }
        }

        public function logEvent(entry:String):void
        {
            log.appendText(entry + "\n");
            trace(entry);
        }
    }
}
```

Flex-exempel

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    invoke="onInvoke(event)" title="Invocation Event Log">
    <mx:Script>
    <![CDATA[
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;

    public function onInvoke(invokeEvent:InvokeEvent):void {
        var now:String = new Date().toTimeString();
        logEvent("Invoke event received: " + now);

        if (invokeEvent.currentDirectory != null){
            logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
        } else {
            logEvent("--no directory information available--");
        }

        if (invokeEvent.arguments.length > 0){
            logEvent("Arguments: " + invokeEvent.arguments.toString());
        } else {
            logEvent("--no arguments--");
        }
    }

    public function logEvent(entry:String):void {
        log.text += entry + "\n";
        trace(entry);
    }
    ]]>
    </mx:Script>
    <mx:TextArea id="log" width="100%" height="100%" editable="false"
        valueCommit="log.verticalScrollPosition=log.textHeight;"/>
</mx:WindowedApplication>
```

Anropa ett AIR-program när användaren loggar in

Adobe AIR 1.0 och senare

Du kan ange att ett AIR-program ska startas automatiskt när den aktuella användaren logga in genom att ange egenskapen `startAtLogin` för NativeApplication till `true`. När du har gjort det startar programmet automatiskt när användaren loggar in. Det fortsättar att starta vid inloggning tills inställningen ändras till `false`, tills användaren ändrar inställningen manuellt via operativsystemet eller tills programmet avinstalleras. Att starta vid inloggning är en körningsinställning. Inställningen gäller bara för den aktuella användaren. Programmet måste vara installerat för att du ska kunna ställa in egenskapen `startAtLogin` på `true`. Ett fel returneras om egenskapen ställs in när ett program inte är installerat (till exempel när det startas med ADL).

Obs! *Programmet startas inte när datorsystemet startas. Det startas när användaren loggar in.*

Om du vill veta om ett program har startats automatiskt eller om användaren har startat det kan du titta på egenskapen `reason` för InvokeEvent-objektet. Om egenskapen är lika med `InvokeEventReason.LOGIN` har programmet startats automatiskt. För andra anropssökvägar ställs egenskapen `reason` in enligt följande:

- `InvokeEventReason.NOTIFICATION` (endast iOS) – Programmet anropades via APN:er. Mer information om APN:er finns i Använda push-meddelanden.
- `InvokeEventReason.OPEN_URL` – Programmet anropades från annat program eller system.
- `InvokeEventReason.Standard` – Alla andra fall.

För att komma åt egenskapen `reason` måste AIR 1.5.1 eller senare vara mål för ditt program (genom att du anger korrekt namnutrymmesvärde i programbeskrivningsfilen).

Följande förenklade program använder egenskapen reason för InvokeEvent för att fastställa hur det ska bete sig när en invoke-händelse inträffar. Om egenskapen reason är "login" ligger programmet kvar i bakgrunden. I annat fall visas huvudprogrammet för användaren. Ett program som använder det här mönstret startas vanligen vid inloggning så att det kan köra bakgrundsprocesser eller händelseövervakning, och ett fönster öppnas som svar på en invoke-händelse från användaren.

```
package {
    import flash.desktop.InvokeEventReason;
    import flash.desktop.NativeApplication;
    import flash.display.Sprite;
    import flash.events.InvokeEvent;

    public class StartAtLogin extends Sprite
    {
        public function StartAtLogin()
        {
            try
            {
                NativeApplication.nativeApplication.startAtLogin = true;
            }
            catch ( e:Error )
            {
                trace( "Cannot set startAtLogin:" + e.message );
            }

            NativeApplication.nativeApplication.addEventListener( InvokeEvent.INVOKE, onInvoke );
        }

        private function onInvoke( event:InvokeEvent ):void
        {
            if( event.reason == InvokeEventReason.LOGIN )
            {
                //do background processing...
                trace( "Running in background..." );
            }
            else
            {
                this.stage.nativeWindow.activate();
            }
        }
    }
}
```

Obs! *Om du vill se skillnaden i beteende paketerar och installerar du programmet. Det går bara att ange egenskapen* `startAtLogin` *för installerade program.*

Anropa ett AIR-program från webbläsaren

Adobe AIR 1.0 och senare

Med hjälp av webbläsarens anropsfunktion kan en webbplats starta ett installerat AIR-program från webbläsaren. Webbläsaranrop tillåts bara om programbeskrivningsfilen ställer in `allowBrowserInvocation` på `true`:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

När programmet anropas via webbläsaren skickar programmets NativeApplication-objekt ett BrowserInvokeEvent-objekt.

Om du vill ta emot BrowserInvokeEvent-händelser, måste du anropa metoden `addEventListener()` för NativeApplication-objektet (`NativeApplication.nativeApplication`) i AIR-programmet. När en händelseavlyssnare registrerar sig för en BrowserInvokeEvent-händelse, kommer den också att ta emot alla BrowserInvokeEvent-händelser som har inträffat före registreringen. Dessa händelser skickas efter att anropet `addEventListener()` returneras, men inte nödvändigtvis före andra BrowserInvokeEvent-händelser som tas emot efter registreringen. Detta innebär att du kan hantera BrowserInvokeEvent-händelser som har inträffat innan din initieringskod verkställs (som när programmet anropas från webbläsaren). Kom ihåg, att om du lägger till en händelseavlyssnare senare i körningen (efter programinitieringen) kommer den fortfarande att få alla BrowserInvokeEvent-händelser som har inträffat sedan programmet startade.

BrowserInvokeEvent-objektet har följande egenskaper:

Egenskap	**Beskrivning**
argument	En array med argument (strängar) som ska skickas till programmet.
isHTTPS	Anger om innehållet i webbläsaren använder https-URL-schemat (`true`) eller inte (`false`).
isUserEvent	Anger om webbläsaranropet resulterar i en användarhändelse (t.ex. ett musklick). I AIR 1.0 är detta alltid inställt på `true`. AIR kräver en användarhändelse för webbläsaranropsfunktionen.
sandboxType	Sandlådetypen för innehållet i webbläsaren. Giltiga värden är samma som dem som kan användas i egenskapen `Security.sandboxType`, och kan vara något av följande: • `Security.APPLICATION` – Innehållet finns i programmets säkerhetssandlåda. • `Security.LOCAL_TRUSTED` – Innehållet finns i lokal-med-filsystems säkerhetssandlåda. • `Security.LOCAL_WITH_FILE` – Innehållet finns i lokal-med-filsystems säkerhetssandlåda. • `Security.LOCAL_WITH_NETWORK` – Innehållet finns i lokal-med-nätverkstjänsts säkerhetssandlåda. • `Security.REMOTE` – Innehållet är i en fjärrdomän (nätverk).
securityDomain	Säkerhetsdomänen för innehållet i webbläsaren som t.ex. `"www.adobe.com"` eller `"www.example.org"`. Den här egenskapen ställs bara in för innehåll i fjärrsäkerhetssandlådan (för innehåll från en nätverksdomän). Den ställs inte in för innehåll i en lokal sandlåda eller en av typen application.

Om du använder funktionen för webbläsaranrop måste du tänka på säkerheten. När en webbplats startar ett AIR-program kan den skicka data via egenskapen `arguments` för objektet BrowserInvokeEvent. Var försiktig när du använder dessa data i känsliga åtgärder, till exempel API:er för fil- eller kodinläsning. Risknivån beror på vad programmet gör med informationen. Om du förväntar dig att bara en viss webbplats ska anropa programmet bör programmet kontrollera egenskapen `securityDomain` för objektet BrowserInvokeEvent. Du kan också kräva att den webbplats som anropar programmet använder HTTPS, vilket du kan verifiera genom att kontrollera egenskapen `isHTTPS` för objektet BrowserInvokeEvent.

Programmet bör verifiera de data som skickas. Om till exempel ett program förväntas att skicka URL:er till en viss domän, bör det verifiera att dessa URL:er verkligen pekar på den domänen. Detta förhindrar att en angripare kan lura programmet att skicka känsliga data.

Inget program bör använda BrowserInvokeEvent-argument som pekar på lokala resurser. Ett program bör till exempel inte skapa File-objekt som baseras på en sökväg som skickas från webbläsaren. Om fjärrsökvägen förväntas bli skickad från webbläsaren, bör programmet se till att sökvägarna inte använder protokollet `file://` i stället för ett fjärrprotokoll.

Avsluta program

Adobe AIR 1.0 och senare

Det snabbaste sättet att avsluta ett program är att anropa metoden NativeApplication exit(). Det fungerar utmärkt när programmet inte behöver spara data eller rensa externa resurser. När du anropar `exit()` stängs alla fönster och sedan avslutas programmet. Om du vill tillåta att fönster eller andra komponenter i programmet avbryter avslutningsprocessen, kanske för att spara viktiga data, ska du skicka rätt varningshändelser innan `exit()` anropas.

Något annat som du behöver tänka på när det gäller att stänga ett program är att tillhandahålla en enda körningssökväg, oavsett hur avstängningsprocessen startar. Användaren (eller operativsystemet) kan utlösa programavslut på följande sätt:

- Genom att stänga det sista programfönstret när `NativeApplication.nativeApplication.autoExit` är `true`.
- Genom att välja programavslutskommandot från operativsystemet, till exempel när användaren väljer programavslutskommandot på standardmenyn. (Detta händer bara i Mac OS. I Windows och Linux går det inte att avsluta programmet via systemets fönsterstandard.)
- Genom att stänga av datorn.

När ett avslutskommando skickas via operativsystemet utmed någon av dessa vägar, skickar NativeApplication en `exiting`-händelse. Om ingen avlyssnare avbryter `exiting`-händelsen stängs alla öppna fönster. Varje fönster skickar en `closing`-händelse och sedan en `close`-händelse. Om något av fönstren avbryter `closing`-händelsen stannar avstängningsprocessen.

Om den ordning som fönstren stängs i är viktig för programmet, lyssnar du efter `exiting`-händelsen från NativeApplication och stänger fönstren själv i rätt ordning. Du kan till exempel behöva göra det om du har ett dokumentfönster med verktygspaletter. Det kan vara opraktiskt eller ännu värre om systemet stänger paletterna, men användaren bestämmer sig för att avbryta avslutskommandot för att spara data. Den enda gången som du får en `exiting`-händelse i Windows är efter att du har stängt det sista fönstret (när egenskapen `autoExit` för NativeApplication-objektet ställs in på `true`).

Om du vill att beteendet ska vara detsamma på alla plattformar, oavsett om avslutssekvensen sätts igång via operativsystemet, menykommandon eller programlogik, bör du utgå från följande metodtips för att avsluta programmet:

1. Skicka alltid en `exiting`-händelse via objektet NativeApplication innan du anropar `exit()` i programkoden och kontrollera att ingen annan komponent i programmet avbryter händelsen.

```
public function applicationExit():void {
    var exitingEvent:Event = new Event(Event.EXITING, false, true);
    NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        NativeApplication.nativeApplication.exit();
    }
}
```

2. Lyssna efter programmets `exiting`-händelse från objektet `NativeApplication.nativeApplication` och stäng alla fönster i hanteraren (skicka först en `closing`-händelse). Utför de rensningsuppgifter som behövs, till exempel spara programdata eller ta bort tillfälliga filer, när alla fönster har stängts. Använd bara synkrona metoder under rensningen så att du är säker på att de är klara innan programmet avslutas.

   Om det inte spelar någon roll i vilken ordning fönstren stängs, kan du göra en slinga genom `NativeApplication.nativeApplication.openedWindows`-arrayen och stänga fönstren i tur och ordning. Om ordningen *spelar roll* tillhandahåller du ett sätt att stänga fönstren i rätt ordning.

```
private function onExiting(exitingEvent:Event):void {
    var winClosingEvent:Event;
    for each (var win:NativeWindow in NativeApplication.nativeApplication.openedWindows) {
        winClosingEvent = new Event(Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 Fönster ska alltid hantera sin egen upprensning genom att avlyssna sina egna `closing`-händelser.

4 Använd bara en `exiting`-avlyssnare i programmet eftersom hanterare som anropas tidigare inte kan veta om efterföljande hanterare kommer att avbryta `exiting`-händelsen (och det vore dumt att lita på körningsordningen).

Kapitel 51: Arbeta med körtids- och operativsystemsinformation i AIR

Adobe AIR 1.0 och senare

I detta avsnitt beskrivs hur ett AIR-program kan hantera filassociationer i operativsystemet, identifiera användaraktivitet och hämta information om Adobe® AIR®-miljön.

Fler hjälpavsnitt

flash.desktop.NativeApplication

Hantera filassociationer

Adobe AIR 1.0 och senare

Associationer mellan ditt program och en filtyp måste deklareras i programbeskrivningen. Under installationsprocessen associerar AIR-programinstalleraren AIR-programmet som standardprogram för att öppna filer för var och en av de deklarerade filtyperna, om inte ett annat program redan är standard. Installationsprocessen för AIR-program skriver inte över befintliga filtypsassociationer. Anropa metoden `NativeApplication.setAsDefaultApplication()` vid körning för att ta över en association från ett annat program.

Det är god praxis att verifiera att den förväntade filassociationen är på plats när programmet startar. Detta beror på att AIR-programinstalleraren inte åsidosätter befintliga filassociationer, och att filassociationer på en användares system kan ändras när som helst. När ett annat program har den aktuella filassociationen anses det också som god ton att fråga användaren innan den befintliga associationen tas över.

Följande metoder i klassen NativeApplication gör att ett program kan hantera filassociationer. Var och en av dessa metoder tar emot filtypstillägget som en parameter:

Metod	Beskrivning
isSetAsDefaultApplication()	Returnerar true om AIR-programmet för tillfället är associerat till den angivna filtypen.
setAsDefaultApplication()	Skapar associationen mellan AIR-programmet och öppna-åtgärden för filtypen.
removeAsDefaultApplication()	Tar bort associationen mellan AIR-programmet och filtypen.
getDefaultApplication()	Rapporterar sökvägen till det program som just nu är associerat med filtypen.

AIR kan bara hantera associationer för de filtyper som ursprungligen deklarerades i programbeskrivningen. Du kan inte hämta information om associationer för icke-deklarerade filtyper, även om användaren har skapat en association mellan filtypen och ditt program manuellt. Ett anrop till någon av metoderna för filassociationshantering med ett tillägg för en filtyp som inte är deklarerad i programbeskrivningen gör att programmet utlöser ett körtidsundantag.

Hämta körtidsversion och korrigeringsnivå

Adobe AIR 1.0 och senare

NativeApplication-objektet har en `runtimeVersion`-egenskap, vilket är den version av körtiden i vilken programmet körs (en sträng som till exempel `"1.0.5"`). NativeApplication-objektet har också en `runtimePatchLevel`-egenskap, vilket är körtidens korrigeringsnivå (en siffra, som till exempel 2960). Följande kod använder dessa egenskaper:

```
trace(NativeApplication.nativeApplication.runtimeVersion);
trace(NativeApplication.nativeApplication.runtimePatchLevel);
```

Identifiera AIR-funktioner

Adobe AIR 1.0 och senare

För en fil som är paketerad med Adobe AIR-programmet är egenskapen `Security.sandboxType` satt till det värde som definieras av konstanten `Security.APPLICATION`. Du kan läsa in innehåll (som både kan innehålla eller sakna API:er som är specifika för AIR) baserat på om en fil ligger i säkerhetssandlådan för Adobe AIR eller ej, vilket illustreras i följande kod:

```
if (Security.sandboxType == Security.APPLICATION)
{
    // Load SWF that contains AIR APIs
}
else
{
    // Load SWF that does not contain AIR APIs
}
```

Alla resurser som inte installeras med AIR-programmet tilldelas samma säkerhetssandlådor som de skulle ha tilldelats av Adobe® Flash® Player i en webbläsare. Fjärresurser placeras i sandlådor baserat på deras källdomäner. Lokala resurser placeras i sandlådorna lokal-med-nätverk, lokal-med-filsystem eller lokal-tillförlitlig.

Du kan kontrollera att den statiska egenskapen `Capabilities.playerType` är satt till `"Desktop"` för att se om innehåll körs i körtiden (och inte körs i Flash Player som körs i en webbläsare).

Mer information finns under ”AIR-säkerhet” på sidan 1081.

Spåra användarnärvaro

Adobe AIR 1.0 och senare

NativeApplication-objektet genererar två händelser som hjälper dig att detektera när en användare aktivt använder datorn. Om ingen mus- eller tangentbordsaktivitet detekteras under det intervall som bestäms av egenskapen `NativeApplication.idleThreshold`, genererar NativeApplication-objektet en `userIdle`-händelse. När nästa tangentbords- eller musinmatning inträffar, genererar NativeApplication-objektet en `userPresent`-händelse. Intervallet för `idleThreshold` mäts i sekunder och har ett standardvärde på 300 (fem minuter). Du kan också hämta det antal sekunder som gått sedan den senaste användarinmatningen från egenskapen `NativeApplication.nativeApplication.lastUserInput`.

Följande kodrader ställer in inaktivitetströskeln till två minuter och lyssnar efter både `userIdle`- och `userPresent`-händelser.

```
NativeApplication.nativeApplication.idleThreshold = 120;
NativeApplication.nativeApplication.addEventListener(Event.USER_IDLE, function(event:Event) {
    trace("Idle");
});
NativeApplication.nativeApplication.addEventListener(Event.USER_PRESENT,
function(event:Event) {
    trace("Present");
});
```

Obs! *Endast en `userIdle`-händelse genereras mellan två `userPresent`-händelser.*

Kapitel 52: Arbeta med inbyggda fönster i AIR

Adobe AIR 1.0 och senare

Du använder klasserna i Adobe® AIR®-API:t för inbyggda fönster när du skapar och hanterar skrivbordsfönster.

Grunderna om interna fönster i AIR

Adobe AIR 1.0 och senare

Följande snabbstartartiklar på Adobe Developer Connection innehåller korta förklaringar och kodexempel på hur du arbetar med inbyggda fönster i AIR:

- Skapa ett program med genomskinliga fönster (Flex)
- Interagera med fönster (Flex)
- Anpassa utseendet på inbyggda fönster (Flex)
- Öppna fönster (Flex)
- Skapa enkla meddelandefönster (Flex)
- Ange fönstrens visningsordning (Flex)
- Skapa icke-rektangulära fönster som kan storleksändras (Flex)
- Interagera med fönster (Flash)
- Anpassa utseendet på inbyggda fönster (Flash)
- Skapa enkla meddelandefönster (Flash)
- Ange fönstrens visningsordning (Flash)
- Skapa icke-rektangulära fönster som kan storleksändras (Flash)

AIR innehåller ett lättanvänt fönster-API för flera plattformar som används för att skapa inbyggda operativsystemsfönster med Flash®-, Flex™- och HTML-programmeringstekniker.

Med AIR får du stor flexibilitet i utvecklingen av programmets utseende. Fönstren som du skapar kan efterlikna ett vanligt skrivbordsprogram, matcha Apple-stilen när programmet körs på en Mac-dator, följa Microsofts konventioner när det körs på en Windows-dator och harmonisera med fönsterhanteraren i Linux – utan en enda rad plattformsspecifik kod. Du kan också använda den utökningsbara fönsterstandarden från Flex Framework med stöd för skal om du vill skapa en egen stil som är oberoende av var programmet körs. Du kan till och med skapa en egen fönsterstandard med vektor- och bitmappsgrafik som har fullständigt stöd för genomskinlighet och alfablandning på skrivbordet. Trött på rektangulära fönster? Rita ett runt fönster.

Fönster i AIR

Adobe AIR 1.0 och senare

AIR stöder tre olika API:er för arbete med fönster:

- Den ActionScript-orienterade NativeWindow-klassen tillhandahåller ett fönster-API på den lägsta nivån. Använd NativeWindow i ActionScript och i program skapade med Flash Professional. Överväg att utöka klassen NativeWindow för att anpassa fönstren som används i ditt program.
- I HTML-miljön kan du använda JavaScript-klassen Window på samma sätt som i ett webbläsarbaserat webbprogram. Anrop till Window-metoder i JavaScript vidarebefordras till det underliggande inbyggda fönsterobjektet.
- mx:WindowedApplication- och mx:Window-klasserna i Flex-ramverket inkluderar en Flex-wrapper för NativeWindow-klassen. WindowedApplication-komponenten ersätter Application-komponenten när du skapar ett AIR-program med Flex och måste alltid användas som det första fönstret i Flex-programmet.

ActionScript-fönster

När du skapar fönster med klassen NativeWindow använder du Flash Player-scenen och visningslistan direkt. Om du vill lägga till ett visuellt objekt i en NativeWindow-instans lägger du till objektet i fönsterscenens visningslista eller i en annan behållare för visningsobjekt på scenen.

HTML-fönster

När du skapar HTML-fönster använder du HTML, CSS och JavaScript för att visa innehållet. Om du vill lägga till ett visuellt objekt i ett HTML-fönster lägger du till innehållet i HTML DOM. HTML-fönster är en specialkategori i NativeWindow. AIR-värden definierar en `nativeWindow`-egenskap i HTML-fönster som ger åtkomst till den underliggande NativeWindow-instansen. Du kan använda den här egenskapen när du vill komma åt de egenskaper, metoder och händelser för NativeWindow som beskrivs här.

Obs! *JavaScript-objektet Window innehåller också metoder för skript i innehållsfönstret, till exempel* `moveTo()` *och* `close()`*. Där det finns överlappande metoder kan du använda den metod som du tycker passar bäst.*

Flex Framework-fönster

När du skapar fönster med Flex Framework använder du vanligen MXML-komponenter för att fylla fönstret med innehåll. Om du vill lägga till en Flex-komponent i ett fönster lägger du till komponentelementet i fönstrets MXML-definition. Du kan också använda ActionScript om du vill lägga till innehåll dynamiskt. Komponenterna mx:WindowedApplication och mx:Window har utformats som Flex-behållare och kan därför hantera Flex-komponenter direkt, vilket inte NativeWindow-objekt kan. Vid behov kan du komma åt egenskaper och metoder för NativeWindow via objekten WindowedApplication och Window med hjälp av egenskapen `nativeWindow`.

Det inledande programfönstret.

Programmets första fönster skapas automatiskt åt dig av AIR. Fönstrets egenskaper och innehåll anges med hjälp av de parametrar som har angetts i elementet `initialWindow` i programbeskrivningsfilen.

Om rotinnehållet är en SWF-fil skapar AIR en NativeWindow-instans, läser in SWF-filen och lägger till den på fönsterscenen. Om rotinnehållet är en HTML-fil skapar AIR ett HTML-fönster och läser in HTML-innehållet.

Klasser för inbyggda fönster

Adobe AIR 1.0 och senare

API:t för inbyggda fönster innehåller följande klasser:

Paket	Klasser
flash.display	• NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	• NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

Händelseflöde för inbyggda fönster

Adobe AIR 1.0 och senare

Inbyggda fönster skickar händelser för att meddela berörda komponenter att en viktig ändring kommer att utföras eller redan har utförts. Många fönsterrelaterade händelser skickas i par. Den första händelsen varnar om att en ändring kommer att utföras. Den andra händelsen meddelar att ändringen har utförts. Du kan avbryta en varningshändelse, men inte en meddelandehändelse. I följande sekvens illustreras det händelseflöde som inträffar när en användare klickar på maximeringsknappen i ett fönster:

1. NativeWindow-objektet skickar en `displayStateChanging`-händelse.
2. Om händelsen inte avbryts av några registrerade lyssnare maximeras fönstret.
3. NativeWindow-objektet skickar en `displayStateChange`-händelse.

   Dessutom skickar NativeWindow-objektet också händelser för relaterade ändringar av fönstrets storlek och position. Fönstret skickar inga varningshändelser för de här relaterade ändringarna. De relaterade händelserna är:

   a. En `move`-händelse skickas om fönstrets övre, vänstra hörn flyttas på grund av maximeringsåtgärden.
   b. En `resize`-händelse skickas om fönstrets storlek ändras på grund av maximeringsåtgärden.

   Ett NativeWindow-objekt skickar en liknande händelsesekvens när ett fönster minimeras, återställs, stängs, flyttas eller storleksändras.

   Varningshändelserna skickas endast när en ändring initieras via fönsterstandardelement eller andra operativsystemsstyrda mekanismer. När du anropar en Window-metod för att ändra fönstrets storlek, position eller visningstillstånd skickar fönstret endast en händelse för att meddela ändringen. Om du vill kan du skicka en varningshändelse med Window-metoden `dispatchEvent()` och sedan kontrollera om varningshändelsen har avbrutits innan du fortsätter med ändringen.

   Detaljerad information om klasserna, metoderna, egenskaperna och händelserna i fönster-API:t finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Egenskaper som styr stil och funktion för inbyggda fönster

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande egenskaper styr grundläggande utseende och funktioner för ett fönster:

- `type`
- `systemChrome`
- `transparent`
- `owner`

När du skapar ett fönster anger du de här egenskaperna i NativeWindowInitOptions-objektet som skickas till fönsterkonstruktorn. Egenskaperna för det inledande programfönstret läses från programbeskrivningen. (Förutom egenskapen `type`, som inte kan anges i programbeskrivningen och därför alltid anges till `normal`.) Egenskaperna kan inte ändras efter att fönstret har skapats.

En del inställningar för de här egenskaperna utesluter varandra: `systemChrome` kan inte anges till `standard` när antingen `transparent` är `true` eller `type` är `lightweight`.

Fönstertyper

Adobe AIR 1.0 och senare

Fönstertyperna i AIR kombinerar attribut för fönsterstandard och synlighet i det underliggande operativsystemet för att skapa tre olika funktionstyper för fönster. Använd de konstanter som finns definierade i klassen NativeWindowType för att referera till typnamnen i koden. AIR innehåller följande fönstertyper:

Typ	Beskrivning
Normal	Ett standardfönster. Normala fönster använder fönsterstandard i fullstorleksstil och visas i Aktivitetsfältet i Windows och på Mac OS X-fönstermenyn.
Verktyg	En verktygspalett. Verktygsfönster använder en tunnare version av systemfönsterstandarden och visas inte i Aktivitetsfältet i Windows eller på Mac OS X-fönstermenyn.
Enkel	Enkla fönster har inga fönsterstandardelement och visas inte i Aktivitetsfältet i Windows eller på Mac OS X-fönstermenyn. Dessutom innehåller enkla fönster inte System-menyn (Alt+Blanksteg) i Windows. Enkla fönster är lämpliga för meddelanderutor och kontroller som kombinationsrutor som öppnas en kort tid i visningsområdet. När `type` är lightweight måste `systemChrome` anges till `none`.

Fönsterstandard

Adobe AIR 1.0 och senare

Fönsterstandard är den uppsättning kontroller som gör att användarna kan ändra ett fönster i skrivbordsmiljön. Fönsterstandardelementen inkluderar namnlisten, knapparna i namnlisten, kanten och handtagen för att ändra storlek.

Systemfönsterstandard

Du kan ange egenskapen `systemChrome` till `standard` eller `none`. Välj fönsterstandarden `standard` om du vill att fönstret ska få en uppsättning standardkontroller som skapas och formges av användarens operativsystem. Välj `none` om du vill ange en egen fönsterstandard för fönstret. Använd de konstanter som finns definierade i klassen NativeWindowSystemChrome för att referera till inställningarna för systemfönsterstandard i koden.

Systemfönsterstandarden hanteras av systemet. Programmet har ingen direkt åtkomst till själva kontrollerna, men kan reagera på händelser som skickas när kontrollerna används. När du använder fönsterstandarden standard för ett fönster måste egenskapen `transparent` anges till `false` och egenskapen `type` måste vara `normal` eller `utility`.

Flex-fönsterstandard

När du använder Flex-komponenterna WindowedApplication eller Window kan fönstret antingen använda systemfönsterstandard eller fönsterstandard från Flex Framework. Om du vill använda Flex-fönsterstandard anger du egenskapen `systemChrome` som används för att skapa fönstret till `none`. Om du använder Flex 4 spark-komponenter i stället för mx-komponenter måste du ange skalklass för att kunna använda Flex-fönsterstandard. Du kan använda de inbyggda skalen eller skapa egna skal. I följande exempel visas hur du använder det inbyggda spark-skalklassen WindowedApplication för fönsterstandard:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
@namespace "library://ns.adobe.com/flex/spark";
WindowedApplication
{
    skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
}
</fx:Style>
</s:WindowedApplication>
```

Mer information finns i Using Flex 4: About the AIR window containers: Controlling window chrome

Anpassad fönsterstandard

När du skapar ett fönster utan systemfönsterstandard måste du lägga till egna fönsterstandardkontroller som hanterar interaktionen mellan användaren och fönstret. Du kan också skapa genomskinliga, icke-rektangulära fönster.

Om du vill använda anpassad fönsterstandard med mx:WindowedApplication eller mx:Window components, måste du ange `showFlexChrome`-formatet till `false`. Annars används den inbyggda fönsterstandarden i Flex.

Fönstergenomskinlighet

Adobe AIR 1.0 och senare

Om du vill tillåta alfablandning för ett fönster med skrivbordet eller andra fönster anger du egenskapen `transparent` för fönstret till `true`. Egenskapen `transparent` måste anges innan fönstret skapas och kan inte ändras.

Ett genomskinligt fönster har ingen standardbakgrund. Alla fönsterområden som inte innehåller något objekt som har ritats av programmet är osynliga. Om ett visat objekt har en alfainställning som är lägre än ett är objektet genomskinligt och allt under objektet syns, till exempel andra fönster, skrivbordet och andra visningsobjekt i samma fönster.

Genomskinliga fönster är användbara när du vill skapa program med kanter som har oregelbunden form eller som ”tonar ut” eller verkar vara osynliga. Tänk på att det kan ta lång tid att återge stora alfablandade områden och använd effekten med måtta.

Viktigt! *I Linux skickas inte mushändelser genom helt genomskinliga pixlar. Undvik att skapa fönster med stora, helt genomskinliga områden eftersom de blockerar användarens åtkomst till andra fönster och objekt på datorn. På Mac OS X och Windows skickas mushändelser genom helt genomskinliga pixlar.*

Det går inte att använda genomskinlighet med fönster som använder systemfönsterstandard. Dessutom kan inte SWF- och PDF-innehåll i HTML visas i genomskinliga fönster. Mer information finns i "Att tänka på vid inläsning av SWF- eller PDF-innehåll i en HTML-sida" på sidan 1010.

Den statiska egenskapen `NativeWindow.supportsTransparency` meddelar huruvida fönstergenomskinlighet är tillgängligt. När genomskinlighet inte stöds visas programmet mot en svart bakgrund. I de fallen visas genomskinliga områden av programmet som ogenomskinliga och svarta. Det är lämpligt att ha en reservlösning om egenskapens test blir `false`. Du kan till exempel visa en varningsdialogruta för användaren eller visa ett rektangulärt, ogenomskinligt reservgränssnitt.

Observera att genomskinlighet alltid stöds i operativsystemen Mac och Windows. I Linux-operativsystem krävs en compositing-fönsterhanterare, men även om en sådan är aktiv kanske genomskinlighet inte stöds på grund av användardefinierade visningsalternativ eller maskinvarukonfigurationen.

Genomskinlighet i ett MXML-programfönster

Adobe AIR 1.0 och senare

Som standard är bakgrunden i ett MXML-fönster ogenomskinlig, även om du skapar fönstret som *transparent*. (Notera genomskinlighetseffekten i fönstrets hörn.) Om du vill visa en genomskinlig bakgrund för fönstret anger du en bakgrundsfärg och ett alfavärde i formatmallen eller <mx:Style>-elementet i programmets MXML-fil. Exempelvis ger följande formatdeklaration bakgrunden en något genomskinlig grön ton:

```
WindowedApplication
{
    background-alpha:".8";
    background-color:"0x448234";
}
```

Genomskinlighet i ett HTML-programfönster

Adobe AIR 1.0 och senare

Som standard är bakgrunden för HTML-innehåll som visas i HTML-fönster och HTMLLoader-objekt ogenomskinlig, även om innehållsfönstret är genomskinligt. Om du vill inaktivera den standardbakgrund som visas för HTML-innehåll anger du egenskapen `paintsDefaultBackground` till `false`. I följande exempel skapas ett HTMLLoader-objekt och standardbakgrunden inaktiveras:

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

I det här exemplet används JavaScript för att inaktivera standardbakgrunden i ett HTML-fönster:

```
window.htmlLoader.paintsDefaultBackground = false;
```

Om ett element i HTML-dokumentet anger en bakgrundsfärg är bakgrunden för det elementet inte genomskinlig. Det går inte att ange ett värde för delvis genomskinlighet (eller ogenomskinlighet). Du kan däremot använda en genomskinlig grafik i PNG-format som bakgrund för en sida eller ett sidelement och uppnå en liknande visuell effekt.

Fönsterägarskap

Ett fönster kan *äga* ett eller flera andra fönster. Dessa ägda fönster visas alltid framför huvudfönstret, minimeras och återställs tillsammans med huvudfönstret och stängs när huvudfönstret stängs. Fönsterägarskap kan inte överföras till ett annat fönster eller tas bort. Ett fönster kan bara ägas av ett huvudfönster, men kan i sin tur äga flera andra fönster.

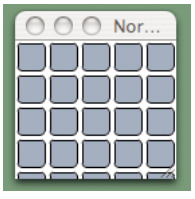
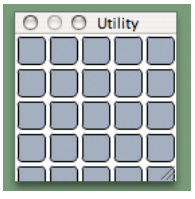
Du kan använda fönsterägarskap för att underlätta hanteringen av fönster som används för verktygspaletter och dialogrutor. Om du till exempel visar en Spara-dialogruta som är kopplad till ett dokumentfönster kan du låta dokumentfönstret äga dialogrutan så att dialogrutan automatiskt visas framför dokumentfönstret.

- NativeWindow.owner
- Christian Cantrell: Owned windows in AIR 2.6

En visuell fönsterkatalog

Adobe AIR 1.0 och senare

Följande tabell visar vilka visuella effekter som olika kombinationer av fönsteregenskaper får på operativsystemen Mac OS X, Windows och Linux:

Fönsterinställningar	Mac OS X	Microsoft Windows	Linux*
type: normal systemChrome: standard transparent: false			
type: utility systemChrome: standard transparent: false			

Fönsterinställningar	Mac OS X	Microsoft Windows	Linux*
type: Valfritt systemChrome: none transparent: false			
type: Valfritt systemChrome: none transparent: true			
mxWindowedApplication eller mx:Window type: Valfritt systemChrome: none transparent: true	mx:Win...	mx:Window	mx:Window

**Ubuntu med Compiz-fönsterhanteraren*

***Obs!** Följande element från systemfönsterstandarden stöds inte i AIR: verktygsfältet i Mac OS X, proxyikonen i OS X, namnlistikonerna i Windows och alternativ systemfönsterstandard.*

Skapa fönster

Adobe AIR 1.0 och senare

Det första fönstret för ett program skapas automatiskt i AIR, men du kan själv skapa alla ytterligare fönster du behöver. Om du vill skapa ett inbyggt fönster använder du konstruktormetoden NativeWindow.

Om du vill skapa ett HTML-fönster använder du antingen HTMLLoader-metoden `createRootWindow()` eller så anropar du JavaScript-metoden `window.open()` från ett HTML-dokument. Fönstret som skapas är ett NativeWindow-objekt vars visningslista innehåller ett HTMLLoader-objekt. HTMLLoader-objektet tolkar och visar HTML- och JavaScript-innehåll i fönstret. Du får åtkomst till egenskaperna för det underliggande NativeWindow-objektet från JavaScript med egenskapen `window.nativeWindow`. (Den här egenskapen är endast tillgänglig för kod som körs i AIR-programsandlådan.)

När du initierar ett fönster, inklusive det inledande programfönstret, kan det vara bra att först skapa fönstret i osynligt läge och sedan läsa in innehållet eller uppdatera det grafiska innehållet innan du gör fönstret synligt. På detta sätt undviker du att eventuella ryckiga visuella ändringar visas för användarna. Du kan ange att det inledande fönstret för programmet ska skapas i osynligt läge med taggen `<visible>false</visible>` i programbeskrivningen (du behöver inte ha med taggen överhuvudtaget eftersom `false` är standardvärdet). Nya NativeWindows-fönster är osynliga som standard. När du skapar ett HTML-fönster med HTMLLoader-metoden `createRootWindow()` kan du ange argumentet `visible`till `false`. Anropa NativeWindow-metoden `activate()` eller ange egenskapen `visible` till `true` om du vill göra ett fönster synligt.

Ange initieringsegenskaper för fönster

Adobe AIR 1.0 och senare

Det går inte att ändra initieringsegenskaperna för ett inbyggt fönster efter att skrivbordsfönstret har skapats. De här oföränderliga egenskaperna och deras standardvärden är bland andra:

Egenskap	Standardvärde
systemChrome	standard
type	normal
transparent	false
owner	null
maximizable	true
minimizable	true
resizable	true

Ange egenskaperna för det inledande fönster som skapas av AIR i programbeskrivningsfilen. Huvudfönstret i ett AIR-program har alltid typen *normal*. (Det går att ange ytterligare fönsteregenskaper i beskrivningsfilen, till exempel `visible`, `width` och `height`, men de här egenskaperna kan ändras när som helst.)

Ange egenskaperna för andra inbyggda fönster och HTML-fönster som skapas av programmet med hjälp av klassen NativeWindowInitOptions. När du skapar ett fönster måste du skicka ett NativeWindowInitOptions-objekt som anger fönstrets egenskaper för antingen konstruktorfunktionen NativeWindow eller HTMLLoader-metoden `createRootWindow()`.

Med följande kod skapas ett NativeWindowInitOptions-objekt för ett verktygsfönster:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.type = NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

Det går inte att ange *systemChrome* till *standard* när *transparent* är `true` eller `type` är `lightweight`.

Obs! *Du kan inte ange initieringsegenskaperna för ett fönster som har skapats med JavaScript-funktionen `window.open()`. Du kan däremot åsidosätta hur de här fönstren skapas genom att implementera en egen HTMLHost-klass. Mer information finns i* *"Hantera JavaScript-anrop till window.open()" på sidan 1021.*

När du skapar ett fönster med Flex-klassen mx:Window anger du initieringsegenskaperna i själva fönsterobjektet, antingen i fönstrets MXML-deklaration eller i koden som fönstret skapas med. Det underliggande NativeWindow-objektet skapas inte förrän du anropar metoden `open()`. När ett fönster har öppnats går det inte att ändra de här initieringsegenskaperna.

Skapa det inledande programfönstret

Adobe AIR 1.0 och senare

I AIR skapas det inledande programfönstret baserat på de egenskaper som anges i programbeskrivningen, och filen som refereras till i innehållselementet läses in. Innehållselementet måste referera till en SWF- eller HTML-fil.

Det inledande fönstret kan fungera som huvudfönster för programmet eller bara som ett sätt att öppna ett eller flera ytterligare fönster. Det behöver inte vara synligt överhuvudtaget.

Skapa det inledande fönstret med ActionScript

Adobe AIR 1.0 och senare

När du skapar ett AIR-program med ActionScript måste huvudklassen för programmet utöka Sprite-klassen (eller en underklass till Sprite). Den här klassen fungerar som huvudstartpunkt för programmet.

När programmet startas skapas ett fönster och sedan en instans av huvudklassen som därefter läggs till på fönsterscenen. Om du vill komma åt fönstret kan du lyssna efter händelsen `addedToStage` och sedan använda egenskapen `nativeWindow` för objektet Stage för att hämta en referens till objektet NativeWindow.

I följande exempel illustreras den grundläggande strukturen för huvudklassen i ett AIR-program som har skapats med ActionScript:

```
package {
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;

    public class MainClass extends Sprite
    {
        private var mainWindow:NativeWindow;
        public function MainClass(){
            this.addEventListener(Event.ADDED_TO_STAGE, initialize);
        }

        private function initialize(event:Event):void{
            mainWindow = this.stage.nativeWindow;
            //perform initialization...
            mainWindow.activate(); //show the window
        }
    }
}
```

Obs! *I praktiken går det att få åtkomst till `nativeWindow`-egenskapen genom konstruktorfunktionen för huvudklassen. Men detta är ett specialfall och gäller endast för det inledande programfönstret.*

När du skapar ett program i Flash Professional skapas huvuddokumentklassen automatiskt om du inte skapar din egen i en separat ActionScript-fil. Du får åtkomst till NativeWindow-objektet för det inledande fönstret genom `nativeWindow`-egenskapen för scenen. Följande kod aktiverar till exempel huvudfönstret i maximerat tillstånd (från tidslinjen):

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

Skapa det inledande fönstret med Flex

Adobe AIR 1.0 och senare

När du skapar ett AIR-program med Flex Framework använder du mx:WindowedApplication som rotelement i MXML-huvudfilen. (Du kan använda komponenten mx:Application, men den har inte stöd för alla funktioner som är tillgängliga i AIR.) Komponenten WindowedApplication fungerar som programmets startpunkt.

När du startar programmet skapas ett inbyggt fönster, Flex Framework initieras och objektet WindowedApplication läggs till på fönsterscenen. När startsekvensen har slutförts skickar WindowedApplication en `applicationComplete`-händelse. Du kommer åt skrivbordsfönsterobjektet med egenskapen `nativeWindow` för WindowedApplication-instansen.

I följande exempel skapas en enkel WindowedApplication-komponent som anger x- och y-koordinaterna:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    applicationComplete="placeWindow()">
    <mx:Script>
        <![CDATA[
            private function placeWindow():void{
                this.nativeWindow.x = 300;
                this.nativeWindow.y = 300;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

Skapa en NativeWindow-instans

Adobe AIR 1.0 och senare

Om du vill skapa en NativeWindow-instans skickar du ett NativeWindowInitOptions-objekt till NativeWindow-konstruktorn:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow:NativeWindow = new NativeWindow(options);
```

Fönstret visas inte förrän du anger egenskapen `visible` till `true` eller anropar metoden `activate()`.

När fönstret har skapats kan du initiera dess egenskaper och läsa in innehåll i fönstret med hjälp av scenegenskapen och tekniker för visningslistan i Flash.

I nästan alla fall ska du ange scenegenskapen `scaleMode` för ett nytt inbyggt fönster till `noScale` (använd konstanten `StageScaleMode.NO_SCALE`). Skalningslägena i Flash har utformats för situationer där programförfattaren inte känner till proportionerna för programmets visningsområde i förväg. Med hjälp av skalningslägena kan författaren välja den kompromiss som gör minst skada: beskära, sträcka ut eller trycka ihop innehållet eller fylla ut med tomt utrymme. Eftersom du anger visningsområdet i AIR (fönsterramen) kan du anpassa fönstret efter innehållet eller innehållet efter fönstret utan att behöva kompromissa.

Skalningsläget för Flex- och HTML-fönster anges automatiskt till `noScale`.

Obs! *Du kan identifiera de största och minsta tillåtna fönsterstorlekarna i det aktuella operativsystemet genom att använda följande statiska NativeWindow-egenskaper:*

```
var maxOSSize:Point = NativeWindow.systemMaxSize;
var minOSSize:Point = NativeWindow.systemMinSize;
```

Skapa ett HTML-fönster

Adobe AIR 1.0 och senare

Om du vill skapa ett HTML-fönster kan du antingen anropa JavaScript-metoden `Window.open()` eller så kan du anropa metoden `createRootWindow()` för klassen HTMLLoader i AIR.

HTML-innehåll i säkerhetssandlådor kan använda JavaScript-standardmetoden `Window.open()`. Om innehållet körs utanför programmets säkerhetssandlåda kan metoden `open()` endast anropas som svar på användaråtgärder, till exempel en musklickning eller tangenttryckning. När `open()` anropas skapas ett fönster med systemfönsterstandard för att visa innehållet på den angivna URL:en. Till exempel:

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

Obs! *Du kan utöka klassen HTMLHost i ActionScript för att anpassa det fönster som skapades med JavaScript-funktionen `window.open()`. Se även "Om att utöka klassen HTMLHost" på sidan 1014.*

Innehåll i programmets säkerhetssandlåda har åtkomst till den kraftfullare metoden för att skapa fönster, `HTMLLoader.createRootWindow()`. Med den här metoden kan du ange alla alternativ för att skapa ett nytt fönster. Med JavaScript-koden nedan skapas exempelvis ett fönster av enkel typ utan systemfönsterstandard med storleken 300×400 pixlar:

```
var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

Obs! *Om innehållet som läses in av ett nytt fönster finns utanför programmets säkerhetssandlåda saknar Window-objektet följande AIR-egenskaper: `runtime`, `nativeWindow` eller `htmlLoader`.*

Om du skapar ett genomskinligt fönster visas inte alltid SWF-innehåll som är inbäddat i HTML som läses in i det fönstret. Du måste ställa in parametern `wmode` för objektet eller den inbäddade taggen som används som referens för SWF-filen på antingen `opaque` eller `tranparent`. Standardvärdet för `wmode` är `window`, så som standard visas inte SWF-innehåll i genomskinliga fönster. PDF-innehåll kan inte visas i genomskinliga fönster, oavsett vilket `wmode` som är angivet. (Före AIR 1.5.2. kunde inte SWF-innehåll visas i genomskinliga fönster heller.)

Fönster som skapas med metoden `createRootWindow()` förblir oberoende av öppningsfönstret. Egenskaperna `parent` och `opener` för JavaScript-objektet Window är `null`. Öppningsfönstret kan komma åt Window-objektet för det nya fönstret med hjälp av HTMLLoader-referensen som returneras av funktionen `createRootWindow()`. I sammanhanget för det föregående exemplet skulle programsatsen `newHTMLLoader.window` referera till JavaScript-objektet Window för det fönster som skapas.

Obs! *Funktionen `createRootWindow()` kan anropas både från JavaScript och ActionScript.*

Skapa ett mx:Window

Adobe AIR 1.0 och senare

Om du vill skapa ett mx:Window kan du skapa en MXML-fil med mx:Window som rottagg eller anropa Window-klassens konstruktor direkt.

I följande exempel skapas och visas ett mx:Window genom ett anrop till Window-konstruktorn:

```
 var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

Lägga till innehåll i ett fönster

Adobe AIR 1.0 och senare

Hur du lägger till innehåll i ett AIR-fönster beror på fönstrets typ. Med exempelvis MXML och HTML kan du definiera fönstrets grundläggande innehåll med hjälp av en deklaration. Du kan bygga in resurser i programmets SWF-filer eller läsa in dem från separata programfiler. Flex-, Flash- och HTML-innehåll kan skapas direkt och läggas till dynamiskt i ett fönster.

När du läser in SWF-innehåll, eller HTML-innehåll som innehåller JavaScript, måste du ta hänsyn till säkerhetsmodellen i AIR. Allt innehåll i programmets säkerhetssandlåda, alltså innehåll som har installerats med programmet och kan läsas in med schemat app: URL, har fullständig åtkomst till alla API:er i AIR. Innehåll som läses in från en plats utanför den här säkerhetssandlådan har inte åtkomst till API:erna i AIR. JavaScript-innehåll utanför programmets säkerhetssandlåda kan inte använda egenskaperna `runtime`, `nativeWindow` eller `htmlLoader` för JavaScript-objektet Window.

Du kan möjliggöra säker korsskriptning genom att använda en sandlådebrygga som tillhandahåller ett begränsat gränssnitt mellan innehåll i och utanför programmet. I HTML-innehåll kan du också mappa sidor från programmet till en säkerhetssandlåda utanför programmet, så att korsskriptning med externt innehåll kan utföras för koden på sidorna. Se även ”AIR-säkerhet” på sidan 1081.

Läsa in en SWF-fil eller bild

Du kan läsa in Flash SWF-filer eller bilder till ett inbyggt fönsters visningslista med klassen `flash.display.Loader`:

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.display.Loader;

    public class LoadedSWF extends Sprite
    {
        public function LoadedSWF(){
            var loader:Loader = new Loader();
            loader.load(new URLRequest("visual.swf"));
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);
        }

        private function loadFlash(event:Event):void{
            addChild(event.target.loader);
        }
    }
}
```

Obs! *Äldre SWF-filer som har skapats med ActionScript 1 eller 2 delar globala tillstånd som klassdefinitioner, singleton-objekt och globala variabler om de läses in i samma fönster. Om en sådan SWF-fil är beroende av oförändrade globala tillstånd för att fungera korrekt går det inte att läsa in den mer än en gång i samma fönster. Det går inte heller att läsa in den i samma fönster som en annan SWF-fil som använder överlappande klassdefinitioner och variabler. Det här innehållet kan läsas in i separata fönster.*

Läsa in HTML-innehåll i en NativeWindow-instans

Om du vill läsa in HTML-innehåll i en NativeWindow-instans kan du antingen lägga till ett HTMLLoader-objekt på fönsterscenen och läsa in HTML-innehållet i HTMLLoader-objektet eller skapa ett fönster som redan innehåller ett HTMLLoader-objekt genom att använda metoden `HTMLLoader.createRootWindow()`. I följande exempel visas HTML-innehåll i ett visningsområde på 300×500 pixlar på scenen för ett inbyggt fönster:

```
//newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

Om du vill läsa in en HTML-sida i ett Flex-program kan du använda HTML-komponenten för Flex.

SWF-innehåll i en HTML-fil visas inte om genomskinlighet används för fönstret (egenskapen `transparent` för fönstret är `true`) såvida parametern `wmode` för objektet eller den inbäddade taggen som används som referens för SWF-filen inte är inställd på antingen`opaque` eller `transparent`. Eftersom standardvärdet för `wmode` är `window` visas som standard inte SWF-innehåll i ett genomskinligt fönster. PDF-innehåll visas inte i ett genomskinligt fönster oavsett vilket `wmode`-värde som används.

Varken SWF- eller PDF-innehåll visas om HTMLLoader-kontrollen är skalad eller roterad, eller om egenskapen `alpha` är inställd på ett annat värde än 1.0.

Lägga till SWF-innehåll som ett överlägg i ett HTML-fönster

Eftersom HTML-fönster är innehåll i en NativeWindow-instans kan du lägga till Flash-visningsobjekt både över och under HTML-lagret i visningslistan.

Om du vill lägga till en visningsobjekt över HTML-lagret använder du metoden `addChild()` för egenskapen `window.nativeWindow.stage`. Metoden `addChild()` lägger till innehåll i lager över befintligt innehåll i fönstret.

Om du vill lägga till en visningsobjekt under HTML-lagret använder du metoden `addChildAt()` för egenskapen `window.nativeWindow.stage` och skickar värdet noll för parametern `index`. När ett objekt placeras vid indexvärdet noll flyttas befintligt innehåll, inklusive HTML-visningen, ett lager uppåt och nytt innehåll infogas längst ned. För att innehåll i lager under HTML-sidan ska synas måste du ange egenskapen `paintsDefaultBackground` för objektet `HTMLlLoader` till `false`. Dessutom kommer inga element på sidan för vilken en bakgrundsfärg har angetts att vara genomskinliga. Om du exempelvis anger en bakgrundsfärg för sidans body-element kommer ingen del av sidan att vara genomskinlig.

I följande exempel illustreras hur du lägger till Flash-visningsobjekt som överlägg och underlägg på en HTML-sida. I exemplet skapas två enkla Shape-objekt, varav det ena läggs till under HTML-innehållet och det andra över. I exemplet uppdateras även Shape-placeringen baserat på händelsen `enterFrame`.

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
        this.shape.x += this.vX;
        this.shape.y += this.vY;

        //Keep the sprite within the window
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
```

```
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);

    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

I det här exemplet ges en grundläggande introduktion till några avancerade tekniker som överträder gränserna mellan JavaScript och ActionScript i AIR. Om du är obekant med att använda visningsobjekt i ActionScript, se ”Visningsprogrammering” på sidan 151 i *Utvecklarhandbok för ActionScript 3.0*.

Exempel: Skapa ett inbyggt fönster

Adobe AIR 1.0 och senare

I följande exempel illustreras hur ett inbyggt fönster skapas.

```
public function createNativeWindow():void {
    //create the init options
    var options:NativeWindowInitOptions = new NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = NativeWindowSystemChrome.STANDARD;
    options.type = NativeWindowType.NORMAL;

    //create the window
    var newWindow:NativeWindow = new NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    newWindow.stage.align = StageAlign.TOP_LEFT;
    newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;

    //activate and show the new window
    newWindow.activate();
}
```

Hantera fönster

Adobe AIR 1.0 och senare

Du använder egenskaperna och metoderna i klassen NativeWindow för att hantera utseendet, funktionerna och livscykeln för skrivbordsfönster.

Obs! *Det är oftast bättre att hantera fönsterbeteendet med hjälp av framework-klasser när du använder Flex-ramverket. De flesta NativeWindow-egenskaperna och -metoderna är tillgängliga via klasserna mx:WindowedApplication och mx:Window.*

Hämta en NativeWindow-instans

Adobe AIR 1.0 och senare

Om du ska kunna ändra ett fönster måste du först hämta fönsterinstansen. Du kan hämta en fönsterinstans från någon av följande platser:

- Den konstruktor för inbyggda fönster som användes för att skapa fönstret:

  ```
  var win:NativeWindow = new NativeWindow(initOptions);
  ```

- `nativeWindow`-egenskapen för fönsterscenen:

  ```
  var win:NativeWindow = stage.nativeWindow;
  ```

- `stage`-egenskapen för ett visningsobjekt i fönstret:

  ```
  var win:NativeWindow = displayObject.stage.nativeWindow;
  ```

- `target`-egenskapen för en händelse för inbyggda fönster som skickas av fönstret:

  ```
  private function onNativeWindowEvent(event:NativeWindowBoundsEvent):void
  {
      var win:NativeWindow = event.target as NativeWindow;
  }
  ```

- `nativeWindow`-egenskapen för en HTML-sida som visas i fönstret:

  ```
  var win:NativeWindow = htmlLoader.window.nativeWindow;
  ```

- `activeWindow`- och `openedWindows`-egenskaperna för NativeApplication-objektet:

  ```
  var nativeWin:NativeWindow = NativeApplication.nativeApplication.activeWindow;
  var firstWindow:NativeWindow = NativeApplication.nativeApplication.openedWindows[0];
  ```

  `NativeApplication.nativeApplication.activeWindow` refererar till det aktiva fönstret för ett program (men returnerar `null` om det aktiva fönstret inte är ett fönster för det här AIR-programmet).

  `NativeApplication.nativeApplication.openedWindows`-arrayen innehåller alla fönster i ett AIR-program som inte har stängts.

Eftersom Flex-objekten mx:WindowedApplication och mx:Window är visningsobjekt kan du enkelt referera till programfönstret i en MXML-fil med hjälp av egenskapen `stage`, enligt följande:

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

Obs! *Innan komponenten WindowedApplication eller Window har lagts till på fönsterscenen av Flex Framework har egenskapen `stage` för komponenten värdet `null`. Det här fungerar på samma sätt som med komponenten Flex Application, men innebär också att det inte går att komma åt scenen eller NativeWindow-instansen i lyssnare för händelser som inträffar tidigare i initieringscykeln för komponenterna WindowedApplication och Window, till exempel `creationComplete`. Det går att komma åt scenen och NativeWindow-instansen utan risk när händelsen `applicationComplete` har skickats.*

Aktivera, visa och dölja fönster

Adobe AIR 1.0 och senare

Du aktiverar ett fönster genom att anropa NativeWindow-metoden `activate()`. När ett fönster aktiveras placeras det överst, kommer i fokus för tangentbordet och musen och, om det behövs, görs synligt genom att det återställs eller genom att egenskapen `visible` anges till `true`. Ordningen för övriga fönster i programmet ändras inte när ett fönster aktiveras. När metoden `activate()` anropas skickar fönstret en `activate`-händelse.

Om du vill visa ett dolt fönster utan att aktivera det anger du egenskapen `visible` till `true`. Detta placerar fönstret överst, men tilldelar inte fönstret fokus.

Om du vill dölja ett fönster anger du egenskapen `visible` för fönstret till `false`. När ett fönster döljs inaktiveras visningen av såväl fönstret som eventuella ikoner i Aktivitetsfältet och, i Mac OS X, menyalternativet på Fönster-menyn.

När du ändrar synligheten för ett fönster ändras även synligheten för fönster som ägs av det fönstret. Om du till exempel döljer ett fönster, döljs även alla fönster som ägs av det fönstret.

Obs! *I Mac OS X går det inte att helt dölja ett minimerat fönster som har en ikon som visas i fönsterdelen av Dock. Om egenskapen* `visible` *har angetts till* `false` *för ett minimerat fönster visas fortfarande dockningsikonen för fönstret. Om användaren klickar på ikonen återställs fönstret till ett synligt läge och visas.*

Ändra visningsordningen för fönster

Adobe AIR 1.0 och senare

AIR innehåller flera metoder för att ändra visningsordningen för fönster direkt. Du kan placera ett fönster överst eller underst i visningsordningen och du kan placera ett fönster framför eller bakom ett annat fönster. Samtidigt kan användaren ändra ordningen för fönstren genom att aktivera dem.

Du kan behålla ett fönster framför andra fönster genom att ange egenskapen `alwaysInFront` för det till `true`. Om fler än ett fönster har den här inställningen sorteras de fönstren efter den inbördes visningsordningen, men de hamnar alltid framför fönster som har egenskapen `alwaysInFront` angiven till false.

Fönster i den översta gruppen visas också framför fönster i andra program, även om AIR-programmet inte är aktivt. Eftersom detta kan vara störande för användaren bör `alwaysInFront` endast anges till `true` när detta är nödvändigt och lämpligt. Det kan till exempel vara befogat i följande fall:

- För tillfälliga popup-fönster för kontroller som knappbeskrivningar, popup-listor, anpassade menyer eller kombinationsrutor. Eftersom de här fönstren stängs när de förlorar fokus undviks irritationsmomentet att en användare inte kan visa ett annat fönster.
- För extremt angelägna felmeddelanden och varningar. När en oåterkallelig ändring kan utföras om användaren inte agerar i tid kan det vara befogat att placera ett varningsfönster längst fram. De flesta fel och varningar kan emellertid hanteras i den normala fönstervisningsordningen.
- Enkla meddelandefönster med kort varaktighet.

Obs! *Det finns inga begränsningar i AIR för hur egenskapen* `alwaysInFront` *kan användas, men om ditt program stör en användare i arbetet kommer det troligen att hamna i användarens papperskorg.*

Om ett fönster äger andra fönster ordnas de ägda fönstren alltid framför det. Om du anropar `orderToFront()` eller anger `alwaysInFront` som `true` för ett fönster som äger andra fönster, så ordnas de ägda fönstren om tillsammans med det ägande fönstret framför andra fönster, men de ägda fönstren visas fortfarande framför det ägande huvudfönstret.

Att anropa fönsterordningsmetoder för ägda fönster fungerar normalt bland fönster som ägs av samma huvudfönster, men det kan även ändra ordningen för hela gruppen med ägda fönster i förhållande till fönster utanför gruppen. Om du till exempel anropar `orderToFront()` för ett ägt fönster flyttas både det fönstret, dess ägare och alla andra fönster som ägs av samma ägare längst fram i fönstervisningsordningen.

Klassen NativeWindow innehåller följande egenskaper och metoder för att ange visningsordningen för ett fönster i förhållande till andra fönster:

Medlem	Beskrivning
Egenskapen alwaysInFront	Anger huruvida fönstret visas i den översta gruppen med fönster. I nästan alla fall är `false` den bästa inställningen. Om värdet ändras från `false` till `true` placeras fönstret framför alla andra fönster (men det aktiveras inte). Om värdet ändras från `true` till `false` placeras fönstret bakom fönster som är kvar i den översta gruppen, men fortfarande framför andra fönster. Om egenskapen anges till det aktuella värdet för ett fönster ändras inte fönstervisningsordningen. Inställningen `alwaysInFront` har ingen effekt på fönster som ägs av ett annat fönster.
orderToFront()	Placerar fönstret längst fram.
orderInFrontOf()	Placerar fönstret omedelbart framför ett visst fönster.
orderToBack()	Placerar fönstret bakom andra fönster.
orderBehind()	Placerar fönstret omedelbart bakom ett visst fönster.
activate()	Placerar fönstret längst fram (och gör fönstret synligt och tilldelar det fokus).

***Obs!** Om ett fönster är dolt (`visible` är `false`) eller minimerat händer inget när visningsordningsmetoderna anropas.*

I Linux-operativsystemet tillämpar olika fönsterhanterare olika regler för fönstrens visningsordning:

- I vissa fönsterhanterare visas verktygsfönster alltid framför vanliga fönster.
- I vissa fönsterhanterare visas helskärmsfönster som har en `alwaysInFront`-egenskap med värdet `true` alltid framför andra fönster som också har `alwaysInFront` angivet till `true`.

Stänga ett fönster

Adobe AIR 1.0 och senare

Om du vill stänga ett fönster använder du metoden `NativeWindow.close()`.

När ett fönster stängs tas innehållet bort från fönstret, men om andra objekt har referenser till innehållet förstörs inte innehållsobjekten. Metoden `NativeWindow.close()` körs asynkront och det program fönstret innehåller fortsätter att köras under stängningsprocessen. Metoden close() skickar en close-händelse när åtgärden har slutförts. NativeWindow-objektet är i princip fortfarande giltigt, men för de flesta egenskaper och metoder för ett stängt fönster genereras felet IllegalOperationError vid försök till åtkomst. Du kan inte öppna ett stängt fönster på nytt. Kontrollera egenskapen `closed` för ett fönster om du vill testa huruvida fönstret har stängts. Om du bara vill dölja ett fönster anger du egenskapen `NativeWindow.visible` till `false`.

Om egenskapen `Nativeapplication.autoExit` har värdet `true`, vilket är standardinställningen, avslutas programmet när dess sista fönster stängs.

Alla fönster som har en ägare stängs när ägarfönstret stängs. De ägda fönstren skickar inga closing-händelser och kan därför inte förhindra stängningen. En close-händelse skickas.

Tillåta att fönsteråtgärder avbryts

Adobe AIR 1.0 och senare

När ett fönster använder systemfönsterstandard kan användaråtgärder i fönstret avbrytas genom att lämpliga händelser avlyssnas och deras standardfunktioner förhindras. När en användare exempelvis klickar på stängningsknappen för systemfönsterstandarden, skickas händelsen `closing`. Om en registrerad lyssnare anropar metoden `preventDefault()` för händelsen stängs inte fönstret.

När ett fönster inte använder systemfönsterstandard skickas inte automatiskt meddelandehändelser gällande förestående ändringar innan ändringarna utförs. Om du anropar metoderna för att stänga ett fönster, ändrar fönsterstatus eller anger någon av egenskaperna för fönstergränserna går det därför inte att avbryta ändringen. Om du vill skicka meddelanden till komponenter i programmet innan en fönsterändring utförs, kan programlogiken skicka den relevanta meddelandehändelsen med hjälp av metoden `dispatchEvent()` för fönstret.

Exempelvis implementerar följande logik en händelsehanterare som kan avbrytas för en stängningsknapp i ett fönster:

```
public function onCloseCommand(event:MouseEvent):void{
    var closingEvent:Event = new Event(Event.CLOSING,true,true);
    dispatchEvent(closing);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

Metoden `dispatchEvent()` returnerar `false` om metoden `preventDefault()` för händelsen anropas av en lyssnare. Den kan emellertid också returnera `false` av andra orsaker, så det är bättre att uttryckligen använda metoden `isDefaultPrevented()` för att testa huruvida ändringen ska avbrytas.

Maximera, minimera och återställa ett fönster

Adobe AIR 1.0 och senare

Om du vill maximera fönstret använder du NativeWindow-metoden `maximize()`.

```
myWindow.maximize();
```

Om du vill minimera fönstret använder du NativeWindow-metoden `minimize()`.

```
myWindow.minimize();
```

Om du vill återställa fönstret (det vill säga ändra tillbaka till den storlek fönstret hade innan det minimerades eller maximerades) använder du NativeWindow-metoden `restore()`.

```
myWindow.restore();
```

Ett fönster som har en ägare minimeras och återställs när det ägande fönstret minimeras och återställs. Inga händelser skickas av det ägda fönstret när det minimeras på grund av att ägaren minimeras.

Obs! *När ett AIR-fönster maximeras fungerar det inte på det sätt som är standard i Mac OS X. I stället för att växla mellan en programdefinierad "standardstorlek" och den senaste storlek som angavs av användaren, växlar AIR-fönster mellan den senaste storlek som angavs av programmet eller användaren och hela det tillgängliga skärmområdet.*

I Linux-operativsystemet tillämpar olika fönsterhanterare olika regler för fönstrens visningsläge:

- I vissa fönsterhanterare går det inte att maximera verktygsfönster.
- Om en största storlek har angetts för fönstret går det inte att maximera vissa fönster. I andra fönsterhanterare är fönstren maximerade, och det går inte att ändra storlek på dem. Ingen händelse för ändrat visningsläge skickas i något av dessa fall.
- En del fönsterhanterare respekterar inte fönsterinställningarna `maximizable` och `minimizable`.

Obs! *I Linux ändras fönsteregenskaper asynkront. Om du ändrar visningsläget på en rad i programmet och läser värdet på nästa rad, så kommer värdet fortfarande att återspegla den gamla inställningen. På alla plattformar skickar NativeWindow-objektet händelsen `displayStateChange` när visningsläget ändras. Om du behöver utföra någon åtgärd baserat på det nya fönsterläget ska du alltid göra det i en `displayStateChange`-händelsehanterare. Mer information finns i "Avlyssna fönsterhändelser" på sidan 918.*

Exempel: Minimera, maximera, återställa och stänga ett fönster

Adobe AIR 1.0 och senare

Med följande korta MXML-program demonstreras Window-metoderna `maximize()`, `minimize()`, `restore()` och `close()`:

```
<?xml version="1.0" encoding="utf-8"?>

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">


    <mx:Script>
    <![CDATA[
    public function minimizeWindow():void
    {
        this.stage.nativeWindow.minimize();
    }

    public function maximizeWindow():void
    {
        this.stage.nativeWindow.maximize();
    }

    public function restoreWindow():void
    {
        this.stage.nativeWindow.restore();
    }

    public function closeWindow():void
    {
        this.stage.nativeWindow.close();
    }
    ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button label="Minimize" click="minimizeWindow()"/>
        <mx:Button label="Restore" click="restoreWindow()"/>
        <mx:Button label="Maximize" click="maximizeWindow()"/>
        <mx:Button label="Close" click="closeWindow()"/>
    </mx:VBox>

</mx:WindowedApplication>
```

I följande ActionScript-exempel för Flash skapas fyra klickbara textfält som utlöser NativeWindow-metoderna `minimize()`, `maximize()`, `restore()` och `close()`:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    public class MinimizeExample extends Sprite
    {
        public function MinimizeExample():void
        {
            var minTextBtn:TextField = new TextField();
            minTextBtn.x = 10;
            minTextBtn.y = 10;
            minTextBtn.text = "Minimize";
            minTextBtn.background = true;
            minTextBtn.border = true;
            minTextBtn.selectable = false;
            addChild(minTextBtn);
            minTextBtn.addEventListener(MouseEvent.CLICK, onMinimize);

            var maxTextBtn:TextField = new TextField();
            maxTextBtn.x = 120;
            maxTextBtn.y = 10;
            maxTextBtn.text = "Maximize";
            maxTextBtn.background = true;
            maxTextBtn.border = true;
            maxTextBtn.selectable = false;
            addChild(maxTextBtn);
            maxTextBtn.addEventListener(MouseEvent.CLICK, onMaximize);

            var restoreTextBtn:TextField = new TextField();
            restoreTextBtn.x = 230;
            restoreTextBtn.y = 10;
            restoreTextBtn.text = "Restore";
            restoreTextBtn.background = true;
            restoreTextBtn.border = true;
            restoreTextBtn.selectable = false;
            addChild(restoreTextBtn);
            restoreTextBtn.addEventListener(MouseEvent.CLICK, onRestore);

            var closeTextBtn:TextField = new TextField();
            closeTextBtn.x = 340;
            closeTextBtn.y = 10;
            closeTextBtn.text = "Close Window";
            closeTextBtn.background = true;
            closeTextBtn.border = true;
            closeTextBtn.selectable = false;
            addChild(closeTextBtn);
```

```
            closeTextBtn.addEventListener(MouseEvent.CLICK, onCloseWindow);
        }
        function onMinimize(event:MouseEvent):void
        {
            this.stage.nativeWindow.minimize();
        }
        function onMaximize(event:MouseEvent):void
        {
            this.stage.nativeWindow.maximize();
        }
        function onRestore(event:MouseEvent):void
        {
            this.stage.nativeWindow.restore();
        }
        function onCloseWindow(event:MouseEvent):void
        {
            this.stage.nativeWindow.close();
        }
    }
}
```

Ändra storlek på och flytta ett fönster

Adobe AIR 1.0 och senare

När ett fönster använder systemfönsterstandard tillhandahåller systemfönsterstandarden kontroller för att ändra storlek på fönstret och flytta runt det på skrivbordet. Om ett fönster inte använder systemfönsterstandard måste du lägga till egna kontroller för att användaren ska kunna ändra storlek på och flytta fönstret.

Obs! *Om du vill ändra storlek på eller flytta ett fönster måste du först hämta en referens till NativeWindow-instansen. Information om hur du hämtar en fönsterreferens finns i "Hämta en NativeWindow-instans" på sidan 909.*

Ändra storlek på ett fönster

Om du vill att användaren ska kunna ändra storlek på fönstret interaktivt använder du NativeWindow-metoden `startResize()`. När den här metoden anropas från en `mouseDown`-händelse är storleksändringsåtgärden beroende av musen och slutförs när operativsystemet tar emot en `mouseUp`-händelse. När du anropar `startResize()` skickar du ett argument som anger den kant eller det hörn som fönstret ska storleksändras utifrån.

Om du vill ställa in fönsterstorleken programmatiskt anger du fönsteregenskaperna `width`, `height` eller `bounds` till önskade mått. Om du anger bounds (gränser) kan fönstrets storlek och placering ändras samtidigt. Det går emellertid inte att försäkra i vilken ordning ändringarna görs. En del Linux-fönsterhanterare tillåter inte att fönstren överskrider skärmens yttre gränser. Detta innebär att den slutgiltiga fönsterstorleken kan begränsas beroende på i vilken ordning som egenskaperna anges, även om effekten av ändringarna egentligen resulterar i ett giltigt fönster. Om du exempelvis ändrar fönstrets höjd och vågräta placering vid skärmens nedre kant, kanske den fullständiga höjden inte kan tillämpas om höjdändringen tillämpas innan den vågräta placeringen ändras.

Obs! *I Linux ändras fönsteregenskaper asynkront. Om du ändrar storlek på ett fönster på en rad i programmet och läser dimensionerna på en annan rad, kommer de fortfarande att återspegla de gamla inställningarna. På alla plattformar skickar NativeWindow-objektet händelsen `resize` när fönstret ändrar storlek. Om du behöver utföra någon åtgärd, till exempel placera kontroller i fönstret, baserat på den nya storleken eller det nya fönsterläget ska du alltid göra det i en `resize`-händelsehanterare. Mer information finns i "Avlyssna fönsterhändelser" på sidan 918.*

Scenens skalningsläge styr hur fönsterscenen och dess innehåll påverkas när ett fönster storleksändras. Kom ihåg att scenskalningslägena har utformats för situationer, till exempel i en webbläsare, där storleken eller proportionerna på visningsområdet inte styrs av programmet. Oftast får du de bästa resultaten genom att ange egenskapen `scaleMode` för scenen till `StageScaleMode.NO_SCALE`. Om du vill att innehållet i fönstret ska skalas kan du fortfarande ange parametrarna `scaleX` och `scaleY` för innehållet beroende på hur fönstergränserna ändras.

Flytta ett fönster

Om du vill flytta ett fönster utan att ändra storlek på det använder du NativeWindow-metoden `startMove()`. När metoden `startMove()` anropas från en `mouseDown`-händelse är flyttprocessen beroende av musen och slutförs när operativsystemet tar emot en `mouseUp`-händelse, precis som med metoden `startResize()`.

Mer information om metoderna `startResize()` och `startMove()` finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Om du vill flytta ett fönster med hjälp av programmering anger du egenskaperna `x`, `y` eller `bounds` för fönstret till önskad position. Om du anger bounds (gränser) kan både fönstrets storlek och placering ändras samtidigt.

Obs! *I Linux ändras fönsteregenskaper asynkront. Om du flyttar ett fönster på en rad i programmet och läser positionen på nästa rad, så kommer värdet fortfarande att återspegla den gamla inställningen. På alla plattformar skickar NativeWindow-objektet händelsen `move` när placeringen ändras. Om du behöver utföra någon åtgärd baserat på den nya placeringen ska du alltid göra det i en `move`-händelsehanterare. Mer information finns i "Avlyssna fönsterhändelser" på sidan 918.*

Exempel: Ändra storlek på och flytta fönster

Adobe AIR 1.0 och senare

Följande exempel visar hur du initierar storleksändrings- och flyttåtgärder i ett fönster:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.display.NativeWindowResize;

    public class NativeWindowResizeExample extends Sprite
    {
        public function NativeWindowResizeExample():void
        {
            // Fills a background area.
            this.graphics.beginFill(0xFFFFFF);
            this.graphics.drawRect(0, 0, 400, 300);
            this.graphics.endFill();

            // Creates a square area where a mouse down will start the resize.
            var resizeHandle:Sprite =
                createSprite(0xCCCCCC, 20, this.width - 20, this.height - 20);
            resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

            // Creates a square area where a mouse down will start the move.
            var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);
        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite
```

```
        {
            var s:Sprite = new Sprite();
            s.graphics.beginFill(color);
            s.graphics.drawRect(0, 0, size, size);
            s.graphics.endFill();
            s.x = x;
            s.y = y;
            this.addChild(s);
            return s;
        }

        public function onStartResize(event:MouseEvent):void
        {
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);
        }

        public function onStartMove(event:MouseEvent):void
        {
            this.stage.nativeWindow.startMove();
        }
    }
}
```

Avlyssna fönsterhändelser

Adobe AIR 1.0 och senare

Om du vill avlyssna händelser som skickas av ett fönster registrerar du en lyssnare med fönsterinstansen. Om du till exempel vill avlyssna closing-händelsen registrerar du en lyssnare med fönstret enligt följande:

```
myWindow.addEventListener(Event.CLOSING, onClosingEvent);
```

När en händelse skickas refererar egenskapen `target` till det fönster som skickar händelsen.

De flesta fönsterhändelser har två relaterade meddelanden. Det första meddelandet talar om att en fönsterändring kommer att utföras (och kan avbrytas), medan det andra meddelandet talar om att ändringen har utförts. När en användare exempelvis klickar på stängningsknappen i ett fönster, skickas händelsemeddelandet closing. Om inga lyssnare avbryter händelsen stängs fönstret och close-händelsen skickas till eventuella lyssnare.

Vanligtvis skickas varningshändelser, till exempel `closing`, endast när systemfönsterstandarden har använts för att utlösa en händelse. Exempelvis skickas inte `closing`-händelsen automatiskt när metoden `close()` för fönstret anropas – endast `close`-händelsen skickas. Du kan däremot skapa en closing-händelse och skicka den med hjälp av Window-metoden `dispatchEvent()`.

De fönsterhändelser som skickar ett Event-objekt är följande:

Händelse	Beskrivning
activate	Skickas när fönstret får fokus.
deactivate	Skickas när fönstret förlorar fokus.
closing	Skickas när fönstret håller på att stängas. Detta inträffar endast automatiskt när stängningsknappen för systemfönsterstandarden används eller, i Mac OS X, när kommandot Avsluta aktiveras.
close	Skickas när fönstret har stängts.

De fönsterhändelser som skickar ett NativeWindowBoundsEvent-objekt är följande:

Händelse	Beskrivning
moving	Skickas omedelbart innan fönstrets övre, vänstra hörn ändrar position, antingen på grund av flytt, storleksändring eller för att fönstrets visningsläge ändras.
move	Skickas när det övre, vänstra hörnet har ändrat position.
resizing	Skickas omedelbart innan fönstrets bredd eller höjd ändras till följd av antingen storleksändring eller ändring av visningsläget.
resize	Skickar när fönstrets storlek har ändrats.

Med NativeWindowBoundsEvent-händelser kan du använda egenskaperna `beforeBounds` och `afterBounds` för att identifiera fönstergränserna före och efter den förestående eller slutförda ändringen.

De fönsterhändelser som skickar ett NativeWindowDisplayStateEvent-objekt är följande:

Händelse	Beskrivning
displayStateChanging	Skickas omedelbart innan fönstrets visningsläge ändras.
displayStateChange	Skickar när fönstrets visningsläge har ändrats.

Med NativeWindowDisplayStateEvent-händelser kan du använda egenskaperna `beforeDisplayState` och `afterDisplayState` för att identifiera fönstrets visningsläge före och efter den förestående eller slutförda ändringen.

I vissa Linux-fönsterhanterare skickas ingen händelse för ändrat visningsläge när ett fönster som har en inställning för högsta storlek maximeras. (Fönstret visas i maximerat fönsterläge, men fönstrets storlek ändras inte.)

Visa fönster i helskärmsläge

Adobe AIR 1.0 och senare

Om du anger egenskapen `displayState` för Stage till `StageDisplayState.FULL_SCREEN_INTERACTIVE` får fönstret helskärmsläge, och tangentbordsinmatning *är* tillåten i det här läget. (I SWF-innehåll som körs i en webbläsare tillåts inte tangentbordsinmatning.) Användaren avslutar helskärmsläget genom att trycka på ESC.

Obs! *I vissa Linux-fönsterhanterare ändras inte fönstrets storlek för att fylla skärmen om en högsta storlek har angetts för fönstret (men fönsterstandarden tas bort).*

Exempelvis definierar följande Flex-kod ett enkelt AIR-program som konfigurerar en enkel helskärmsterminal:

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to
exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

Följande ActionScript-exempel för Flash simulerar en enkel textterminal i helskärmsläge:

```
    import flash.display.Sprite;
    import flash.display.StageDisplayState;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class FullScreenTerminalExample extends Sprite
    {
        public function FullScreenTerminalExample():void
        {
            var terminal:TextField = new TextField();
            terminal.multiline = true;
            terminal.wordWrap = true;
            terminal.selectable = true;
            terminal.background = true;
            terminal.backgroundColor = 0x00333333;

            this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

            addChild(terminal);
            terminal.width = 550;
            terminal.height = 400;

            terminal.text = "Welcome to the dumb terminal application. Press the ESC key to
exit.\n_";

            var tf:TextFormat = new TextFormat();
            tf.font = "Courier New";
            tf.color = 0x00CCFF00;
            tf.size = 12;
            terminal.setTextFormat(tf);

            terminal.setSelection(terminal.text.length - 1, terminal.text.length);
        }
    }
```

Kapitel 53: Visningsskärmar i AIR

Adobe AIR 1.0 och senare

Använd Screen-klassen i Adobe® AIR® om du vill få åtkomst till information om de visningsskärmar som är anslutna till en dator eller en enhet.

Fler hjälpavsnitt

flash.display.Screen

Grunderna om visningsskärmar i AIR

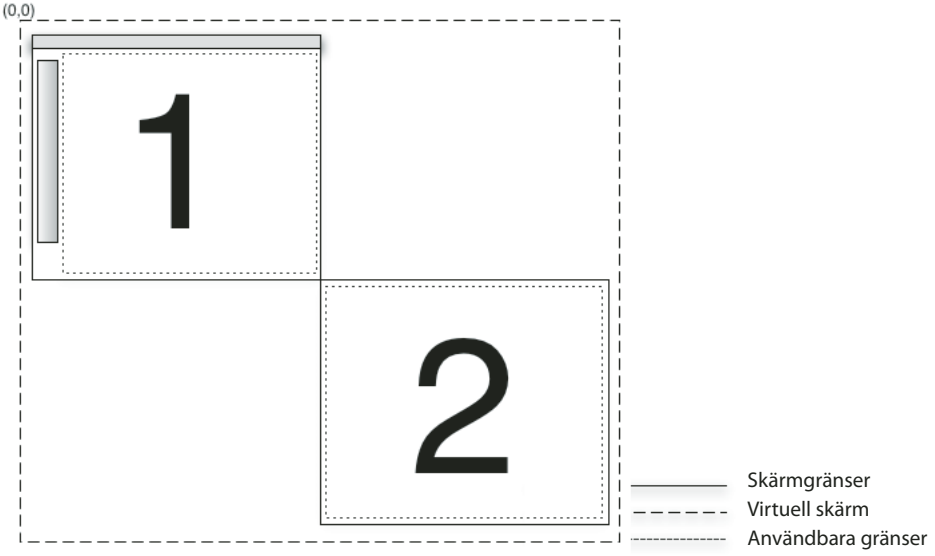
Adobe AIR 1.0 och senare

- Mäta det virtuella skrivbordet (Flex)
- Mäta det virtuella skrivbordet (Flash)

Skärm-API:t innehåller en enda klass, Screen, vilken innehåller statiska medlemmar för att hämta systemskärminformation, och instansmedlemmar för att beskriva en särskild skärm.

Ett datorsystem kan ha flera skärmar anslutna, och dessa kan motsvara flera skrivbordsskärmar som är arrangerade i ett virtuellt utrymme. AIR-klassen Screen tillhandahåller information om skärmarna, deras relativa inställningar och deras användbara utrymme. Om fler än en bildskärm mappar till samma skärm, finns det bara en bildskärm. Om storleken på en skärm är större än bildskärmens visningsområde, finns det inget sätt att avgöra vilken del av skärmen som är synlig för tillfället.

En skärm representerar en oberoende skärmvisningsyta. Skärmar beskrivs som rektanglar inuti det virtuella skrivbordet. Det övre vänstra hörnet på den skärm som är angiven som huvudskärm är origopunkten i det virtuella skrivbordets koordinatsystem. Alla värden som används för att beskriva en skärm ges i pixlar.



I denna skärminställning finns det två skärmar i det virtuella skrivbordet. Koordinaterna för det övre vänstra hörnet på huvudskärmen (#1) är alltid (0,0). Om skärminställningarna ändras så att skärm #2 anges som huvudskärm, blir koordinaterna för skärm #1 negativa. Menyer, aktivitetsfält och dockningar räknas inte med när en skärms användbara gränser rapporteras.

Mer information om klassen, metoderna, egenskaperna och händelserna för skärm-API finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Räkna upp skärmarna

Adobe AIR 1.0 och senare

Du kan räkna upp skärmarna för det virtuella skrivbordet med följande skärmmetoder och -egenskaper:

Metod eller egenskap	Beskrivning
Screen.screens	Innehåller en array av Screen-objekt som beskriver de tillgängliga skärmarna. Ordningen i arrayen har ingen betydelse.
Screen.mainScreen	Innehåller ett Screen-objekt för huvudskärmen. I Mac OS X är huvudskärmen den skärm som visar menyfältet. I Windows är huvudskärmen den huvudskärm som anges i systemet.
Screen.getScreensForRectangle()	Innehåller en array av Screen-objekt som beskriver de skärmar som överlappas av den angivna rektangeln. Den rektangel som skickas till metoden anges i pixelkoordinater i det virtuella skrivbordet. Om det inte finns några skärmar som överlappas av rektangeln är arrayen tom. Du kan använda den här metoden för att avgöra vilka skärmar som visar ett fönster.

Spara inte värden som returneras av metoder och egenskaper för klassen Screen. Användaren eller operativsystemet kan ändra de tillgängliga skärmarna och deras inställningar när som helst.

Följande exempel använder skärm-API:t för att flytta ett fönster mellan flera skärmar till följd av att piltangenterna används. Exemplet hämtar arrayen `screens` och sorterar den antingen lodrätt eller vågrätt (beroende på vilken piltangent som användes) för att flytta fönstret till nästa skärm. Koden stegar sedan igenom den sorterade arrayen och jämför varje skärm med koordinaterna för den aktuella skärmen. Exemplet anropar `Screen.getScreensForRectangle()` och skickar in fönstergränserna för att identifiera fönstrets aktuella skärm.

```
package {
    import flash.display.Sprite;
    import flash.display.Screen;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;

    public class ScreenExample extends Sprite
    {
        public function ScreenExample()
        {
                stage.align = StageAlign.TOP_LEFT;
                stage.scaleMode = StageScaleMode.NO_SCALE;

                stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);
        }

        private function onKey(event:KeyboardEvent):void{
            if(Screen.screens.length > 1){
                switch(event.keyCode){
                    case Keyboard.LEFT :
                        moveLeft();
                        break;
                    case Keyboard.RIGHT :
                        moveRight();
                        break;
                    case Keyboard.UP :
                        moveUp();
                        break;
                    case Keyboard.DOWN :
                        moveDown();
                        break;
                }
            }
        }

        private function moveLeft():void{
            var currentScreen = getCurrentScreen();
            var left:Array = Screen.screens;
            left.sort(sortHorizontal);
            for(var i:int = 0; i < left.length - 1; i++){
                if(left[i].bounds.left < stage.nativeWindow.bounds.left){
                    stage.nativeWindow.x +=
                        left[i].bounds.left - currentScreen.bounds.left;
                    stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
                }
            }
        }

        private function moveRight():void{
            var currentScreen:Screen = getCurrentScreen();
            var left:Array = Screen.screens;
            left.sort(sortHorizontal);
            for(var i:int = left.length - 1; i > 0; i--){
                if(left[i].bounds.left > stage.nativeWindow.bounds.left){
                    stage.nativeWindow.x +=
```

```
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveUp():void{
        var currentScreen:Screen = getCurrentScreen();
        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = 0; i < top.length - 1; i++){
            if(top[i].bounds.top < stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function moveDown():void{
        var currentScreen:Screen = getCurrentScreen();

        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = top.length - 1; i > 0; i--){
            if(top[i].bounds.top > stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function sortHorizontal(a:Screen,b:Screen):int{
        if (a.bounds.left > b.bounds.left){
            return 1;
        } else if (a.bounds.left < b.bounds.left){
            return -1;
        } else {return 0;}
    }

    private function sortVertical(a:Screen,b:Screen):int{
        if (a.bounds.top > b.bounds.top){
            return 1;
        } else if (a.bounds.top < b.bounds.top){
            return -1;
        } else {return 0;}
    }

    private function getCurrentScreen():Screen{
        var current:Screen;
        var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
        (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
        return current;
    }
  }
}
```

Kapitel 54: Skriva ut

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash-miljöer (som Adobe® Flash® Player och Adobe® AIR™) kan kommunicera med ett operativsystems utskriftsgränssnitt så att du kan skicka sidor för utskrift till utskriftshanteraren. Varje sida som skickas till utskriftshanteraren kan rymma innehåll som är synligt, dynamiskt eller utanför skärmen för användaren, däribland databasvärden och dynamisk text. Dessutom innehåller egenskaperna för klassen flash.printing.PrintJob värden som bygger på en användares skrivarinställningar, så att du kan formatera sidorna korrekt.

I följande innehåll tillhandahålls strategier för hur du använder metoderna och egenskaperna för klassen flash.printing.PrintJob för att skapa en utskrift, läsa en användares utskriftsinställningar och göra justeringar i en utskrift utifrån information från Flash-miljön och användarens operativsystem.

Grunder för utskrift

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 kan du använda klassen PrintJob för att skapa ögonblicksbilder av visningsinnehåll som ska konverteras till färg-och-papper-representationen i en utskrift. Att ställa in innehåll för utskrift är detsamma som är ställa in det för skärmvisning. Du placerar och storleksändrar elementen för att skapa önskad layout. En utskrift har emellertid vissa egenheter som skiljer den från en skärmlayout. På skrivaren används till exempel andra upplösningar än på datorskärmar. Innehållet på en datorskärm är dynamiskt och kan ändras medan utskrivet innehållet är statiskt. När du planerar en utskrift måste du tänka på begränsningarna vid fast sidstorlek och möjligheten till flersidesutskrift.

Även om dessa skillnader verkar vara uppenbara är det viktigt att tänka på dem när du ställer in utskriften i ActionScript. Korrekt utskrift avgörs av en kombination av de värden som du anger och egenskaperna hos användarens skrivare. Klassen PrintJob omfattar egenskaper som anger de viktiga egenskaperna i användarens skrivare.

Viktiga termer och begrepp

Följande referenslista innehåller viktiga termer om utskrift:

Utskriftshanterare Den del av operativsystemet eller skrivardrivrutinen som spårar de sidor som väntar på utskrift och skickar dem till skrivaren när den är tillgänglig.

Sidorientering Rotationen av det utskrivna innehållet i relation till papperet. Vågrätt (liggande) eller lodrätt (stående).

Utskriftsjobb Den sida eller den uppsättning sidor som är en enskild utskrift.

Skriva ut en sida

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du använder en instans av klassen PrintJob för att hantera utskriften. Använd dessa fyra programsatser efter varandra för att skriva ut en grundsida i Flash Player eller AIR:

- `nytt PrintJob()`: Skapa en ny utskriftsinstans av namnet du anger.
- `PrintJob.start()`: Initierar utskriften för operativsystemet, anropar utskriftsdialogrutan för användaren och fyller i skrivskyddade egenskaper för utskriftsjobbet.
- `PrintJob.addPage()`: Innehåller detaljer om utskriftsinnehållet, däribland Sprite-objektet (och alla underordnade objekt), storleken på utskriftsområdet och om skrivaren skriver ut bilden som en vektor eller bitmapp. Du kan använda successiva anrop till `addPage()` om du vill skriva ut flera sprite över flera sidor.
- `PrintJob.send()`: Skickar sidorna till operativsystemets skrivare.

Ett enskilt utskriftsskript är till exempel (inklusive satserna `package`, `import` och `class` för kompilering):

```
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();
        }
    }
}
```

***Obs!** Det här exemplet ska visa baselementen i ett utskriftsskript och innehåller inte någon felhantering. Om du vill skapa ett skript som svarar rätt på en användare som avbryter en utskrift går du till "Arbeta med undantag och returer" på sidan 928.*

Om du vill rensa ett PrintJob-objekts egenskaper anger du värdet `null` för PrintJob-variabeln (som i `myPrintJob = null`).

Utskriftsåtgärder för Flash och för systemet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom Flash-miljöerna skickar sidor till operativsystemets utskriftsgränssnitt, måste du känna till vilka åtgärder som hanteras av Flash och vilka åtgärder som hanteras av operativsystemets eget utskriftsgränssnitt. Flash-miljöerna kan initiera en utskrift, läsa vissa av skrivarens sidinställningar, skicka innehållet för en utskrift till operativsystemet och verifiera om användaren eller systemet har avbrutit en utskrift. Andra processer, till exempel att visa skrivarspecifika dialogrutor, avbryta en utskrift i utskriftshanteraren eller rapportera skrivarens status, hanteras av operativsystemet. Flash-miljöerna kan också svara om det uppstår problem med att initiera eller formatera en utskrift, men kan bara rapportera tillbaka om vissa egenskaper och villkor från operativsystemets utskriftsgränssnitt. Om du är utvecklare måste din kod kunna svara på dessa egenskaper och villkor.

Arbeta med undantag och returer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du måste kontrollera om metoden `PrintJob.start()` returnerar `true` innan anropen `addPage()` och `send()` körs om användaren har avbrutit utskriften. Ett enkelt sätt att kontrollera om dessa metoder har avbrutits innan du fortsätter är att lägga in dem i en `if`-programsats, på följande vis:

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Om `PrintJob.start()` är `true` har användaren valt Print (eller en Flash-miljö, som Flash Player eller AIR, har initierat ett Print-kommando). Därmed kan metoderna `addPage()` och `send()` anropas.

För att utskriften ska bli lättare att hantera utlöses undantag för metoden `PrintJob.addPage()`, så att du kan fånga upp fel och ge användaren information och alternativ. Om metoden `PrintJob.addPage()` misslyckas kan du anropa en annan funktion eller avbryta den aktuella utskriften. Du fångar upp dessa undantag genom att bädda in `addPage()`-anrop inuti en `try..catch`-programsats som i följande exempel. I exemplet är `[params]` en platshållare för de parametrar som anger det innehåll som ska skrivas ut:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

När utskriften startar kan du lägga till innehåll med hjälp av `PrintJob.addPage()` och se om detta genererar ett undantag (om till exempel användaren har avbrutit utskriften). I så fall kan du lägga till logik i programsatsen `catch` för att förse användaren (eller Flash-miljön) med information och alternativ, men du kan också avbryta den aktuella utskriften. När du har lagt till sidan kan du fortsätta med att skicka sidorna till skrivaren med `PrintJob.send()`.

Om det uppstår något problem när utskriften skickas till skrivaren (om skrivaren till exempel är offline) kan du fånga upp det här undantaget också och tillhandahålla mer information eller fler alternativ (till exempel visa ett textmeddelande eller animera en varning). Du kan till exempel tilldela ny text till ett textfält i en `if..else`-programsats, vilket visas i följande kod:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Ett fungerande exempel finns i ”Exempel på utskrift: skala, beskära och svara” på sidan 937.

Arbeta med sidegenskaper

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När användaren klickar på OK i utskriftsdialogrutan och `PrintJob.start()` returnerar `true`, kan du komma åt egenskaperna som definierats av skrivarens inställningar. De här inställningarna omfattar pappersbredd, pappershöjd (`pageHeight` och `pageWidth`) och innehållets orientering på papperet. Eftersom detta är skrivarinställningar, som inte styrs av Flash-miljön, kan du inte ändra inställningarna. Du kan emellertid använda dem för att justera det innehåll som du skickar till skrivaren för att matcha de aktuella inställningarna. Mer information finns i ”Ställa in storlek, skala och orientering” på sidan 930.

Ställa in vektor- och bitmappsåtergivning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan ställa in utskriften så att alla sidor mellanlagras som vektorgrafik eller bitmappsbilder. I vissa fall skapar vektorutskrift en mindre utskriftsfil och en bättre bild än bitmappsutskrift. Men om innehållet omfattar en bitmappsbild och du vill spara alla alfagenomskinlighets- eller färgeffekter, måste du skriva ut sidan som en bitmappsbild. En icke-PostScript-skrivare konverterar dessutom automatiskt all vektorgrafik till bitmappsbilder.

Du anger en bitmappsutskrift genom att skicka ett PrintJobOptions-objekt som den tredje parametern av `PrintJob.addPage()`.

För Flash Player och AIR-versioner före AIR 2 anger du PrintJobOptions-objektets `printAsBitmap`-parameter till `true`, enligt följande:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Om du inte anger något värde för den tredje parametern kommer standardvärdet att användas, vilket är vektorutskrift.

För AIR 2 och senare versioner använder du egenskapen `printMethod` för PrintJobOptions-objektet om du vill ange utskriftsmetod. Den här egenskapen tar tre värden, som är definierade som konstanter i klassen PrintMethod:

- `PrintMethod.AUTO`: Väljer automatiskt den bästa utskriftsmetoden beroende på det innehåll som ska skrivas ut. Om en sida till exempel består av text väljs metoden för vektorutskrift. Men om en bild med vattenstämpel och alfagenomskinlighet läggs ovanpå texten väljs bitmappsutskrift för att behålla genomskinligheten.
- `PrintMethod.BITMAP`: Tvingar fram bitmappsutskrift oavsett innehåll
- `PrintMethod.VECTOR`: Tvingar fram vektorutskrift oavsett innehåll

Samordna utskriftsprogramsatser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I ActionScript 3.0 begränsas inte ett PrintJob-objekt till en enda bildruta (som i tidigare versioner av ActionScript). Men eftersom operativsystemet visar utskriftsstatusinformation för användaren när användaren har klickat på OK i utskriftsdialogrutan, bör du anropa `PrintJob.addPage()` och `PrintJob.send()` så fort som möjligt och skicka sidor till utskriftshanteraren. En fördröjning med att nå den bildruta som innehåller `PrintJob.send()`-anropet fördröjer utskriftsprocessen.

I ActionScript 3.0 finns det en skripttimeout på 15 sekunder. Tiden mellan varje större programsats i en utskriftssekvens får därför inte överstiga 15 sekunder. 15-sekunderstimeouten tillämpar m.a.o. följande intervall:

- Mellan `PrintJob.start()` och den första `PrintJob.addPage()`
- Mellan `PrintJob.addPage()` och nästa `PrintJob.addPage()`
- Mellan den sista `PrintJob.addPage()` och `PrintJob.send()`

Om något av ovanstående intervall överskrider 15 sekunder kommer nästa anrop till `PrintJob.start()` för PrintJob-instansen att returnera `false`. Nästa `PrintJob.addPage()` för PrintJob-instansen utlöser ett körningsundantag i Flash Player eller AIR.

Ställa in storlek, skala och orientering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I avsnittet ”Skriva ut en sida” på sidan 927 beskrivs ett grundläggande utskriftsjobb, där utskriften direkt återspeglar den utskrivna motsvarigheten till skärmstorleken och placeringen av angiven sprite. Skrivare använder ju olika upplösningar och de kan ha inställningar som påverkar utseendet av den utskrivna spriten negativt.

Flash-miljöerna kan läsa ett operativsystems utskriftsinställningar, men observera att dessa egenskaper är skrivskyddade: även om du kan svara på deras värden kan du inte ställa in dem. Du kan till exempel hitta skrivarens sidstorleksinställning och justera innehållet så att det passar storleken. Du kan också bestämma en skrivares marginalinställningar och sidorientering. Om du vill svara på skrivarinställningarna måste du ange ett utskriftsområde, justera för olikheter mellan en skärms upplösning och en skrivarens punktmått eller transformera innehållet så att det stämmer överens med storleks- och orienteringsinställningarna för användarens skrivare.

Använda rektanglar för utskriftsområdet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoden `PrintJob.addPage()` kan du ange område för den sprite du vill skriva ut. Den andra parametern `printArea`, är i form av ett Rectangle-objekt. Du kan ange ett värde för den här parametern på tre olika sätt:

- Skapa ett Rectangle-objekt med speciella egenskaper och använda rektangeln i anropet `addPage()` på det här sättet:

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- Om du inte redan har specificerat ett Rectangle-objekt kan du göra det inom själva anropet på det här sättet:

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- Om du tänker ange värden för den tredje parametern i anropet `addPage()` men inte vill ange en rektangel, kan du använda `null` för den andra parametern på det här sättet;

```
myPrintJob.addPage(sheet, null, options);
```

Jämföra punkter och pixlar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Rektangelns bredd och höjd är pixelvärden. För en skrivare används punkter som utskriftsmåttenheter. Punkter har fast fysisk storlek (1/72 tum) medan en pixels storlek beror på bildskärmens upplösning. Därför beror konverteringsförhållandet mellan pixlar och punkter på skrivarinställningarna och om spriten är skalad. En oskalad sprite som är 72 pixlar bred skrivs ut en tum bred, där en punkt är lika med en pixel, oavsett skärmens upplösning.

Du kan konvertera tum eller centimeter till twip eller punkter (en twip är 1/20 punkt) med följande ekvivalenser:

- 1 punkt = 1/72 tum = 20 twip
- 1 tum = 72 punkter = 1440 twip
- 1 centimeter = 567 twip

Om du utelämnar parametern `printArea` eller om den skickas felaktigt skrivs hela området för sprite ut.

Skalning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill skalförändra ett Sprite-objekt innan du skriver ut det anger du skalningsegenskaperna (se ”Ändra storlek och skaländra objekt” på sidan 179) innan du anropar metoden `PrintJob.addPage()`, och när utskriften är klar sätter du tillbaka ursprungsvärdena. Sprite-objektets skala har ingenting med egenskapen `printArea` att göra. Det innebär, att om du anger att utskriften ska vara 50 x 50 pixlar, skrivs 2 500 pixlar ut. Om du har skalat Sprite-objektet skrivs samma 2 500 pixlar ut, men Sprite-objektet skrivs ut i den skalade storleken.

Ett exempel finns i ”Exempel på utskrift: skala, beskära och svara” på sidan 937.

Utskrift med liggande eller stående orientering

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Eftersom inställningarna för orientering identifieras av Flash Player och AIR kan du skapa logik i ActionScript som justerar innehållets storlek och rotation som svar på skrivarinställningarna, vilket visas i följande exempel:

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

Obs! *Om du vill läsa systeminställningen för innehållsorientering på pappret måste du importera klassen PrintJobOrientation. Klassen PrintJobOrientation ger konstanta värden som definierar innehållets orientering på papperet. Importera klassen med följande programsats:*

```
import flash.printing.PrintJobOrientation;
```

Svara på sidhöjd och sidbredd

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du använder en strategi som liknar hantering av skrivarens orienteringsinställningar kan du läsa inställningarna för sidhöjd och sidbredd och svara på dem genom att bädda in viss logik i en `if`-programsats. I följande kod finns ett exempel:

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

En sidas marginalinställningar kan dessutom bestämmas genom att sidan och pappersmåtten jämförs, vilket visas i följande exempel:

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Avancerade utskriftstekniker

Adobe AIR 2 och senare

Från och med Adobe AIR 2 innehåller klassen PrintJob ytterligare egenskaper och metoder, och tre ytterligare klasser stöds: PrintUIOptions, PaperSize och PrintMethod. Dessa ändringar tillåter ytterligare utskriftsarbetsflöden och ger författare större kontroll över utskriftsprocessen. Ändringarna omfattar:

- Dialogrutor med utskriftsformat: Både standarddialogrutor och anpassade dialogrutor för utskriftsformat kan visas. Användaren kan ange sidintervall, pappersstorlek, orientering och skalning före utskrift.
- Utskriftsvy: Ett visningsläge som visar pappersstorlek, marginaler och placeringen för sidan kan skapas.
- Begränsad utskrift: Författare kan begränsa utskriftsalternativen, till exempel intervallet för utskriftsbara sidor.
- Kvalitetsalternativ: Författare kan justera utskriftskvaliteten för ett dokument och tillåta användaren att välja upplösning och färgalternativ.
- Flera utskriftssessioner: En enskild PrintJob-instans kan nu användas för flera utskriftssessioner. Program kan tillhandahålla enhetliga inställningar varje gång dialogrutorna med utskriftsformat och utskrift visas.

Ändringar i utskriftsarbetsflöde

Det nya utskriftsarbetsflödet består av följande steg:

- `new PrintJob()`: Skapar en PrintJob-instans (eller återanvänder en befintlig instans). Många nya egenskaper och metoder för PrintJob, till exempel `selectPaperSize()`, är tillgängliga innan utskriftsjobbet påbörjas eller under utskriftsprocessen.
- `PrintJob.showPageSetupDialog()`: (valfritt) Visa dialogrutan för utskriftsformat utan att påbörja ett nytt utskriftsjobb.
- `PrintJob.start()` eller `PrintJob.start2()`: Utöver metoden `start()` används metoden `start2()` för att initiera utskriftsbuffringen. Metoden `start2()` gör det möjligt att välja om dialogrutan för utskrift ska visas och anpassa den om den ska visas.
- `PrintJob.addPage()`: Lägg till innehåll i utskriftsjobbet. Oförändrad från befintlig process.
- `PrintJob.send()` eller `PrintJob.terminate()`: Skicka sidorna till den valda skrivaren eller avsluta utskriftsjobbet utan att skicka. Utskriftsjobb avslutas som svar på ett fel. Även om en PrintJob-instans avbryts kan den fortfarande återanvändas. Oavsett om ett utskriftsjobb skickas till skrivaren eller avbryts bevaras de aktuella utskriftsinställningarna när du återanvänder PrintJob-instansen.

Dialogruta för utskriftsformat

Metoden `showPageSetupDialog()` visar operativsystemets dialogruta för utskriftsformat, om den aktuella miljön stöder det. Kontrollera alltid egenskapen `supportsPageSetupDialog` innan du anropar den här metoden. Här är ett enkelt exempel:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
//check for static property supportsPageSetupDialog of PrintJob class
if (PrintJob.supportsPageSetupDialog) {
    myPrintJob.showPageSetupDialog();
}
```

Metoden kan även anropas med en egenskap för klassen PrintUIOptions för att kontrollera vilka alternativ som ska visas i dialogrutan för utskriftsformat. Minsta och högsta sidnummer kan anges. I följande exempel begränsas utskriften till de tre första sidorna:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
if (PrintJob.supportsPageSetupDialog) {
    var uiOpt:PrintUIOptions = new PrintUIOptions();
    uiOpt.minPage = 1;
    uiOpt.maxPage = 3;
    myPrintJob.showPageSetupDialog(uiOpt);
}
```

Ändra utskriftsinställningar

Inställningarna för en PrintJob-instans kan ändras när som helst sedan den har konstruerats. Detta inkluderar ändrade inställningar mellan `addPage()`-anrop och efter att ett utskriftsjobb har skickats eller avslutats. Vissa inställningar, till exempel egenskapen `printer`, gäller för hela utskriftsjobbet och inte för enskilda sidor. Dessa inställningar måste anges innan `start()` eller `start2()` anropas.

Metoden `selectPaperSize()` kan anropas för att ange standardpappersstorleken i dialogrutorna för utskriftsformat och utskrift. Den kan även anropas under ett utskriftsjobb för att ange pappersstorlek för ett sidintervall. Den anropas med hjälp av konstanter som definieras i klassen `PaperSize`, som i det här exemplet, där kuvertstorlek 10 används:

```
import flash.printing.PrintJob;
import flash.printing.PaperSize;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.selectPaperSize(PaperSize.ENV_10);
```

Använd egenskapen `printer` för att hämta eller ange namnet på skrivaren för det aktuella utskriftsjobbet. Som standard anger namnet på standardskrivaren. Egenskapen `printer` är `null` om det inte finns några skrivare tillgängliga eller om systemet inte stöder utskrift. Om du vill ändra skrivaren hämtar du först listan över tillgängliga skrivare med egenskapen `printers`. Den här egenskapen är en vektoregenskap vars strängelement består av tillgängliga skrivarnamn. Ange egenskapen `printer` för något av dessa strängvärden för att göra den skrivaren aktiv. Det går inte att ändra egenskapen `printer` för ett aktivt utskriftsjobb. Försök att ändra den efter ett anrop till `start()` eller `start2()`, och innan jobbet skickas eller avslutas, misslyckas. Här är ett exempel på hur den här egenskapen anges:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.printer = "HP_LaserJet_1";
myPrintJob.start();
```

Egenskapen `copies` hämtar värdet för antalet kopior som anges i operativsystemets dialogruta för utskrift. Egenskaperna `firstPage` och `lastPage` hämtar sidintervallet. Egenskapen `orientation` hämtar inställningen för pappersorientering. Du kan ange att de här egenskaperna ska åsidosätta värdena från dialogrutan för utskrift. I följande exempel anges dessa egenskaper:

```
import flash.printing.PrintJob;
import flash.printing.PrintJobOrientation;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.copies = 3;
myPrintJob.firstPage = 1;
myPrintJob.lastPage = 3;
myPrintJob.orientation = PrintJobOrientation.LANDSCAPE;
```

Följande skrivskyddade inställningar som är associerade med `PrintJob` ger användbar information om de aktuella skrivarinställningarna:

- `paperArea`: De rektangulära gränserna för skrivarmaterialet, i punkter.
- `printableArea`: De rektangulära gränserna för det utskriftsbara området, i punkter.
- `maxPixelsPerInch`: Den fysiska upplösningen för den aktuella skrivaren, i pixlar per tum.
- `isColor`: Den aktuella skrivarens förmåga att skriva ut i färg (returnerar`true` om den aktuella skrivaren kan skriva ut i färg).

Se ”Utskriftsexempel: utskriftsformat och utskriftsalternativ” på sidan 938.

Exempel på utskrift: flersidesutskrift

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du skriver ut mer än en innehållssida kan du koppla sidorna till olika sprite (i det här fallet `sheet1` och `sheet2`) och sedan använda `PrintJob.addPage()` för varje sprite. Den här tekniken visas i följande kod:

```
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)
                {
                // Respond to error.
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

Exempel på utskrift: skala, beskära och svara

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I vissa fall kanske vill du justera storleken (eller andra egenskaper) på ett visningsobjekt vid utskriften för att anpassa skillnaderna mellan hur det visas på skärmen och hur det ser ut på utskriften på papperet. När du anpassar egenskaperna för ett visningsobjekt innan du skriver ut det (till exempel med egenskaperna `scaleX` och `scaleY`) och objektet skalförändras så att det blir större än den definierade rektangeln för utskriftsområdet, kommer objektet att beskäras. Du vill förmodligen också återställa egenskaperna när sidorna har skrivits ut.

Med följande kod skalförändras måtten för visningsobjektet `txt` (men inte den gröna bakgrunden) och då beskärs textfältet och får den angivna rektangelns mått. Efter utskriften återfår textfältet till sin ursprungliga storlek för visning på skärmen. Om användaren avbryter utskriften från operativsystemets utskriftsdialogruta, ändras innehållet i Flash-miljön och användaren får en varning om att utskriften har avbrutits.

```
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)
                {
                    // Do nothing.
```

```
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            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;
        }
    }
}
```

Utskriftsexempel: utskriftsformat och utskriftsalternativ

Adobe AIR 2 och senare

I följande exempel initieras PrintJob-inställningarna för antal kopior, pappersstorlek (legal) och sidorientering (liggande). Det tvingar dialogrutan Utskriftsformat att visas först och startar sedan utskriftsjobbet genom att visa dialogrutan Skriv ut.

```
package
{
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.printing.PaperSize;
    import flash.printing.PrintUIOptions;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintAdvancedExample extends Sprite
    {
        private var bg:Sprite = new Sprite();
        private var txt:TextField = new TextField();
        private var pj:PrintJob = new PrintJob();
        private var uiOpt:PrintUIOptions = new PrintUIOptions();

        public function PrintAdvancedExample():void
        {
            initPrintJob();
            initContent();
            draw();
            printPage();
        }

        private function printPage():void
        {
            //test for dialog support as a static property of PrintJob class
            if (PrintJob.supportsPageSetupDialog)
            {
                pj.showPageSetupDialog();
            }
            if (pj.start2(uiOpt, true))
            {
                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job terminated";
                pj.terminate();
            }
        }

        private function initContent():void
        {
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();
```

```
            txt.border = true;
            txt.text = "Hello World";
        }

        private function initPrintJob():void
        {
            pj.selectPaperSize(PaperSize.LEGAL);
            pj.orientation = PrintJobOrientation.LANDSCAPE;
            pj.copies = 2;
            pj.jobName = "Flash test print";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

Kapitel 55: Geolocation

Om en enhet har stöd för geopositionering kan du använda ett API för geopositionering för att hämta enhetens aktuella geografiska plats. Om enheten stöder den här funktionen kan du hämta positionsinformation. Informationen innefattar bland annat höjd, precision, riktning, hastighet och en tidsstämpel för när platsen senast ändrades.

Klassen Geolocation skickar `update`-händelser som svar på enhetens positioneringssensor. Händelsen `update` är ett GeolocationEvent-objekt.

Fler hjälpavsnitt

flash.sensors.Geolocation

flash.events.GeolocationEvent

Identifiera ändringar i geopositionering

Använd sensorn för geopositionering genom att instansiera ett Geolocation-objekt och registrera `update`-händelser som skickas. Händelsen `update` är ett Geolocation-händelseobjekt. Händelsen har åtta egenskaper:

- `altitude` – Höjd över havet i meter.
- `heading` – Rörelsens riktning (med avseende på geografisk nord) i heltalsgrader.
- `horizontalAccuracy` – Vågrät precision i meter.
- `latitude` – Latitud i grader.
- `longitude` – Longitud i grader.
- `speed` – Hastigheten i meter per sekund.
- `timestamp` – Det antal millisekunder som vid tiden för händelsen har passerat sedan körningen initierades.
- `verticalAccuracy` – Lodrät precision i meter.

Egenskapen `timestamp` är ett int-objekt. De övriga är Number-objekt.

Det här grundläggande exemplet visar geopositioneringsdata i ett textfält:

```
var geo:Geolocation;
if (Geolocation.isSupported)
{
    geo = new Geolocation();
    geo.addEventListener(GeolocationEvent.UPDATE, updateHandler);
}
else
{
    geoTextField.text = "Geolocation feature not supported";
}
function updateHandler(event:GeolocationEvent):void
{
    geoTextField.text = "latitude: " + event.latitude.toString() + "\n"
            + "longitude: " + event.longitude.toString() + "\n"
            + "altitude: " + event.altitude.toString()
            + "speed: " + event.speed.toString()
            + "heading: " + event.heading.toString()
            + "horizontal accuracy: " + event.horizontalAccuracy.toString()
            + "vertical accuracy: " + event.verticalAccuracy.toString()
}
```

Om du vill använda exemplet måste du först skapa textfältet `geoTextField` och lägga till det i visningslistan innan koden används.

Du kan justera det önskade tidsintervallet för geopositioneringshändelser genom att anropa metoden `setRequestedUpdateInterval()` för Geolocation-objektet. Den här metoden tar en parameter, `interval`, vilken är det uppdateringsintervall som krävs i millisekunder:

```
var geo:Geolocation = new Geolocation();
geo.setRequestedUpdateInterval(10000);
```

Den faktiska tiden mellan uppdateringarna av geopositioneringen kan vara längre eller kortare än det här värdet. Förändringar av uppdateringsintervallet påverkar alla registrerade avlyssnare. Om du inte anropar metoden `setRequestedUpdateInterval()` tar programmet emot uppdateringar beroende på enhetens standardintervall.

Användaren kan förhindra att ett program får åtkomst till data för geopostionering. En användare med iPhone får till exempel en fråga när programmet försöker att hämta data för geopositionering. Användaren kan då neka programmet till att få åtkomst till data för geopositionering. Geolocation-objektet skickar en `status`-händelse när användaren försöker få åtkomst till data för geopositionering som inte är tillgängliga. Geolocation-objektet har också egenskapen `muted`, som är angiven till `true` när sensorn för geopositionering inte är tillgänglig. Geolocation-objektet skickar en `status`-händelse när egenskapen `muted` ändras. I följande kod visas hur du identifierar att data för geopositionering inte är tillgängliga:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.GeolocationEvent;
    import flash.events.MouseEvent;
    import flash.events.StatusEvent;
    import flash.sensors.Geolocation;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class GeolocationTest extends Sprite
    {

        private var geo:Geolocation;
        private var log:TextField;

        public function GeolocationTest()
        {
            super();
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            setUpTextField();

            if (Geolocation.isSupported)
            {
                geo = new Geolocation();
                if (!geo.muted)
                {
                    geo.addEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
                }
                geo.addEventListener(StatusEvent.STATUS, geoStatusHandler);
            }
            else
            {
                log.text = "Geolocation not supported";
            }
        }

        public function geoUpdateHandler(event:GeolocationEvent):void
        {
            log.text = "latitude : " + event.latitude.toString() + "\n";
            log.appendText("longitude : " + event.longitude.toString() + "\n");
        }

        public function geoStatusHandler(event:StatusEvent):void
        {
            if (geo.muted)
                geo.removeEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
            else
                geo.addEventListener(GeolocationEvent.UPDATE, geoStatusHandler);
        }

        private function setUpTextField():void
```

```
        {
            log = new TextField();
            var format:TextFormat = new TextFormat("_sans", 24);
            log.defaultTextFormat = format;
            log.border = true;
            log.wordWrap = true;
            log.multiline = true;
            log.x = 10;
            log.y = 10;
            log.height = stage.stageHeight - 20;
            log.width = stage.stageWidth - 20;
            log.addEventListener(MouseEvent.CLICK, clearLog);
            addChild(log);
        }
        private function clearLog(event:MouseEvent):void
        {
            log.text = "";
        }
    }
}
```

***Obs!** Första generationens iPhone, de som inte har någon GPS-enhet, skickar `update`-händelser endast sporadiskt. På dessa enheter skickar ett Geolocation-objekt initialt en eller två `update`-händelser. Därefter skickas `update`-händelser när märkbara informationsändringar sker.*

Kontrollera stöd för geopositionering

Använd egenskapen `Geolocation.isSupported` för att testa om körningsmiljön kan använda den här funktionen:

```
if (Geolocation.isSupported)
{
    // Set up geolocation event listeners and code.
}
```

För närvarande stöds geopositionering endast på ActionScript-baserade program för iPhone och i Flash Lite 4. Om värdet för `Geolocation.isSupported` är `true` vid körningen betyder det att det finns stöd för geopositionering.

Vissa iPhone-modeller har inte någon GPS-enhet. Dessa modeller använder andra metoder (till exempel triangulering) för att få geopositioneringsdata. För de här modellerna, samt för alla iPhone-telefoner med inaktiverad GPS, kan ett Geolocation-objekt endast skicka en eller två initiala `update`-händelser.

Kapitel 56: Internationalisering av program

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Med paketet flash.globalization är det enklare att skapa internationell programvara som kan anpassas till konventionerna för olika språk och regioner.

Fler hjälpavsnitt

flash.globalization-paket

”Lokalisera program” på sidan 962

Charles Bihis: Want to Localize your Flex/AIR Apps? (Vill du lokalisera dina Flex-/AIR-program?)

Allmänt om internationalisering av program

Termerna globalisering och internationalisering används ibland som synonymer. De flesta definitionerna av dessa termer säger att globalisering avser en kombination av affärsprocesser och tekniska processer, medan internationalisering endast avser tekniska processer.

Här är några definitioner av viktiga termer:

Globalisering En mängd tekniska processer och affärsprocesser som behövs för att förbereda och skapa produkter och företagsaktiviteter globalt. Globalisering består av tekniska aktiviteter så som internationalisering, lokalisering och kulturalisering samt affärsaktiviteter så som produkthantering, finansiell planering, marknadsföring och juridiskt arbete. Globalisering förkortas ibland som *G11n* (vilket står för ”Globalization”, med bokstaven G, 11 bokstäver till, och slutligen bokstaven n). *"Globalisering är vad företag gör."*

Internationalisering En teknisk process för att generalisera en produkt så att den kan hantera flera språk, skript och kulturella konventioner (inklusive valutor, sorteringsregler, siffer- och datumformat med mera) utan att produktens design eller kompilering behöver göras om. Den här processen kan delas in i två uppsättningar aktiviteter: aktivering och lokalisering. Internationalisering kallas ibland även *world-readiness* (världsförberedd) och förkortas ibland med *I18n*. *"Internationalisering är vad tekniker gör."*

Lokalisering Processen att anpassa en produkt eller en tjänst till ett specifikt språk, en specifik kultur och önskat lokalt utseende och känsla. Lokalisering förkortas ibland *L10n*. *"Lokalisering är vad översättare gör."*

Kulturalisering En teknisk process för att utveckla eller anpassa speciella funktioner för en kulturs unika behov. Några exempel är de japanska publiceringsfunktionerna i Adobe InDesign och funktionen för Hanko-stöd i Adobe Acrobat.

En del andra viktiga internationaliseringstermer kan definieras så här:

Teckenuppsättning De tecken som används för ett språk eller en grupp med språk. En teckenuppsättning omfattar nationella tecken, specialtecken (till exempel interpunktionstecken och matematiska symboler), numeriska tecken och kontrolltecken.

Sortering Sorteringen av text i lämplig ordning för ett angivet språkområde.

Språk Ett värde som representerar de språkliga och kulturella konventionerna som används i en geografisk, politisk eller kulturell region (vilket i många fall avser ett enskilt land). En unik språkområdesidentifierare (språkområdes-ID) representerar detta värde. Språkområdes-ID:t används för att hitta en uppsättning språkområdesdata som stöd för det aktuella språkområdet. Detta stöd omfattar bland annat måttenheter samt tolkning och formatering av tal och datum.

Resursbunt En lagrad uppsättning med språkområdesspecifika element som skapas för ett språkområde där ett program ska användas. En resursbunt består vanligtvis av alla textelement i programmets användargränssnitt. I bunten översätts dessa element till rätt språk för det aktuella språkområdet. Den kan även innehålla andra inställningar som ändrar layouten eller beteendet för användargränssnittet för ett visst språkområde. En resursbunt innehåller andra mediatyper eller referenser till andra mediatyper som är språkområdesspecifika.

Översikt över paketet flash.globalization

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Paketet flash.globalization utnyttjar det underliggande operativsystemets stöd för kulturella funktioner. Det gör det enklare att skriva program som följer de kulturella konventionerna för enskilda användare.

Huvudklasserna i paketet omfattar följande:

- Klassen Collator som styr sortering och matchning av strängar
- Klassen CurrencyFormatter som formaterar siffror i valutabeloppssträngar och tolkar valutabelopp och symboler från indatasträngar
- Klassen DateTimeFormatter som formaterar datumvärden
- Klassen LocaleID som hämtar information om ett visst språkområde
- Klassen NumberFormatter som formaterar och analyserar numeriska värden
- Klassen StringTools som hanterar språkområdeskänslig skiftlägeskonvertering av strängar

flash.globalization-paketet och resurslokalisering

Paketet flash.globalization hanterar inte resurslokalisering. Men du kan använda det språkområdes-ID som finns för flash.globalization som nyckelvärde när du hämtar lokaliserade resurser med andra metoder. Du kan till exempel lokalisera programresurser skapade med Flex och klasserna ResourceManager och ResourceBundle. Mer information finns i Lokalisera Flex-program.

Adobe AIR 1.1 innehåller också vissa funktioner som kan underlätta vid lokalisering av AIR-program och som beskrivs närmare i ”Lokalisera AIR-program” på sidan 963.

Grundläggande information om internationalisering av program

I följande steg beskrivs ett vanligt tillvägagångssätt på hög nivå vid internationalisering av ett program med paketet flash.globalization:

1. Avgör eller ange språkområdet.
2. Skapa en instans av en tjänsteklass (Collator, CurrencyFormatter, DateTimeFormatter, NumberFormatter eller StringTools).
3. Sök efter fel och reservlösningar med egenskaperna för lastOperationStatus.
4. Formatera och visa information med språkområdesspecifika inställningar.

Nästa steg är att läsa in och visa strängar och resurser för användargränssnitt som är specifika för språkområdet. Det här steget kan bland annat omfatta sådant som att:

- Använda funktioner för automatisk layout för att ändra storlek på gränssnittet så att strängarna får plats
- Välja rätt teckensnitt och alternativteckensnitt som stöds
- Använda FTE-textmotorn för stöd för andra skriftspråk
- Kontrollera att redigeringsprogram för inmatningsmetod (IME) hanteras korrekt

Söka efter fel och reservlösningar

Tjänsteklasserna för flash.globalization följer alla samma mönster för att identifiera fel. De delar dessutom ett mönster för reservlösningar från ett otillgängligt begärt språkområde till ett som användarens operativsystem har stöd för.

I följande exempel visas hur du söker efter fel och reservlösningar vid instansiering av tjänsteklasser. Varje tjänsteklass har en lastOperationStatus-egenskap som visar om den senaste metoden aktiverade dess fel eller varningar.

```
var nf:NumberFormatter = new NumberFormatter("de-DE");
if(nf.lastOperationStatus != LastOperationStatus.NO_ERROR)
{
    if(nf.lastOperationStatus == LastOperationStatus.USING_FALLBACK_WARNING)
    {
        // perform fallback logic here, if needed
        trace("Warning - Fallback locale ID: " + nf.actualLocaleIDName);
    }
    else
    {
        // perform error handling logic here, if needed
        trace("Error: " + nf.lastOperationStatus);
    }
}
```

I det här exemplet spåras endast ett meddelande om ett alternativt språkområdes-ID används eller om ett fel uppstår. Ditt program kan utföra ytterligare felhantering om det behövs. Du kan till exempel visa ett meddelande för användaren eller tvinga programmet att använda en specifik språkområdesversion som stöds.

Bestämma språkområde

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Ett språkområde identifierar en särskild uppsättning språkliga och kulturella konventioner för ett visst land eller en viss region.

En identifierare av språkområdet kan hanteras säkert som en sträng. Men du kan använda klassen LocaleID för ytterligare information om ett språkområde.

Du skapar ett LocaleID-objekt enligt följande:

```
var locale:LocaleID = new LocaleID("es-MX");
```

När LocaleID-objektet har skapats kan du hämta information om detta språkområdes-ID. Använd metoderna `getKeysAndValues()`, `getLanguage()`, `getRegion()`, `getScript()`, `getVariant()` och `isRightToLeft()` samt egenskapen `name`.

De värden som du får från dessa metoder och egenskaper kan ge annan information om språkområdet som inte kan fås direkt av identifieraren av språkområdet.

När ett program skapar en språkområdeskänslig tjänst, till exempel en datumformaterare, måste det språkområde som ska användas anges. Listan med språkområden varierar mellan olika operativsystem, vilket kan innebära att det begärda språkområdet inte behöver vara tillgängligt.

Flash Player försöker först matcha språkkoden för det språkområde som du har begärt. Sedan görs ett försök att förfina valet av språkområde genom att hitta ett skrivsystem (skript) och en region som matchar. Till exempel:

```
var loc:LocaleID = new LocaleID("es");
trace(loc.getLanguage()); // es
trace(loc.getScript()); // Latn
trace(loc.getRegion()); // ES
```

I det här exemplet hämtar `LocaleID()`-konstruktorn data om det språkområde som bäst matchar språkkoden ”es” för användaren.

Ange språkområdes-ID

Du kan ange det aktuella språkområdet för ett program på flera olika sätt, till exempel genom att:

- Hårdkoda ett enda språkområdes-ID i programmet. Det här är vanligt, men det stöder inte internationalisering av programmet.
- Använda språkområdesinställningarna från användarens operativsystem eller webbläsare, eller andra användarinställningar. Den här tekniken resulterar ofta i de bästa språkområdesinställningarna för användaren, men är inte alltid de rätta. Det finns en risk att operativsystemets inställningar inte överensstämmer med de språkområdesinställningar som användaren faktiskt föredrar. Användaren kanske till exempel använder en delad dator och kan inte ändra operativsystemets språkområdesinställningar.
- Låta användaren välja språkområde från en lista över de språkområden som stöds efter att språkområdet har angetts baserat på användarens inställningar. Den här strategin är vanligtvis det bästa alternativet om programmet har stöd för flera språkområden.

Du kan implementera det tredje alternativet på följande sätt:

1. Hämta en lista över de språkområden som användaren föredrar från användarprofilen, webbläsarinställningarna, operativsysteminställningarna eller en cookies-fil. (Programmet måste själv implementera logiken. En direkt läsning av sådana här inställningar stöds inte av flash.globalization-biblioteket.)
2. Avgör vilka av dessa språkområden som stöds av ditt program och välj den bästa som standard. Använd metoden LocaleID.determinePreferredLocales() för att hitta de bästa språkområdena för en användare baserat på de språkområdesinställningar som användaren föredrar och de språkområden som stöds av operativsystemet.
3. Ge användaren en möjlighet att ändra standardinställningen för språkområde om användaren skulle vilja det.

Begränsningar för andra klasser för språkområden och språk

Med klassen `fl.lang.Locale` kan du ersätta textsträngar baserat på ett språkområde genom att använda resursbuntar med strängvärden. Den här klassen stöder dock inte andra funktioner för internationalisering, som siffror, valutor, datumformatering, sortering, matchning och så vidare. Dessutom är den här klassen endast tillgänglig med Flash Professional.

Du kan också hämta den aktuella språkkodsinställningen för operativsystemet med egenskapen `flash.system.Capabilities.language`. Den här egenskapen hämtar emellertid endast den två tecken långa språkkoden enligt ISO 639-1, inte det kompletta språkområdes-ID:t, och den har endast stöd för en specifik uppsättning språkområden.

Med AIR 1.5 kan du använda egenskapen `flash.system.Capabilities.languages`. Den här egenskapen innehåller en array med de användarens språkinställningar. Den har alltså inte de begränsningar som finns i `Capabilities.language`.

Formatera tal

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Visningsformatet för numeriska värden kan variera mycket mellan olika regioner. Här visas till exempel hur talet 123456.78 formateras för olika språkområden:

Språk	Talformat
en-US (engelska, USA)	-123,456.78
de-DE (tyska, Tyskland)	-123.456,78
fr-FR (franska, Frankrike)	-123 456,78
de-CH (tyska, Schweiz)	-123'456.78
en-IN (engelska, Indien)	-1,23,456.78
Många arabiska språkområden	123,456.78-

Det finns många faktorer som påverkar talformaten, till exempel:

- Avgränsare. Decimalavgränsaren placeras mellan heltalet och delarna av ett tal. Det kan vara en punkt, ett komma eller ett annat tecken. Grupperingsavgränsaren eller tusendelsavgränsaren kan vara en punkt, ett komma, ett hårt blanksteg eller ett annat tecken.
- Grupperingsmönster. Antalet siffror mellan grupperingsavgränsarna till vänster om decimalpunkten kan bestå av två, tre eller fler siffror.
- Indikatorer för negativa tal. Negativa tal kan visas med ett minustecken till vänster eller till höger om talet, eller inom parentes för olika finansiella program. Det negativa talet 19 kan till exempel visas som -19, 19- eller (19).
- Nollor före och efter talet. För vissa kulturella konventioner läggs nollor till före eller efter de visade talen. Värdet 0,17 kan till exempel visas som ,17, 0,17 eller 0,170 med mera.
- Uppsättning siffertecken. Många språk, till exempel hindi, arabiska och japanska använder olika uppsättningar av siffertecken. I flash.globalization-paketet stöds alla uppsättningar siffertecken som motsvarar siffrorna 0–9.

Klassen NumberFormatter utgörs av alla dessa faktorer för formatering av numeriska värden.

Använda klassen NumberFormatter

Klassen NumberFormatter formaterar numeriska värden (av typen int, uint eller Number) enligt konventionerna för ett specifikt språkområde.

I följande exempel visas det lättaste sättet att formatera ett tal med hjälp av egenskaperna för standardformatering från användarens operativsystem:

```
var nf:NumberFormatter = new NumberFormatter(LocaleID.DEFAULT);
trace(nf.formatNumber(-123456.789))
```

Resultatet varierar beroende på användarens språkområdesinställningar och användarinställningar. Om användarens språkområde till exempel är fr-FR blir det formaterade värdet:

-123.456,789

Om du endast vill formatera ett tal för ett särskilt språkområde, oavsett användarens inställningar, anger du det språkområdet specifikt. Till exempel:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
trace(nf.formatNumber(-123456.789));
```

Resultatet i detta fall är:

-123'456.789

Metoden formatNumber() tar Number som en parameter. Klassen NumberFormatter har också en formatInt()-metod som tar en int som invärde och en formatUint()-metod som tar en uint.

Du kan kontrollera formateringslogiken explicit genom att ange egenskaper för klassen NumberFormatter, enligt exemplet nedan:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
nf.negativeNumberFormat = 0;
nf.fractionalDigits = 5;
nf.trailingZeros = true;
nf.decimalSeparator = ",";
nf.useGrouping = false;
trace(nf.formatNumber(-123456.789)); //(123456.78900)
```

I det här exemplet skapas först ett NumberFormatter-objekt och sedan anges:

- att negativa talformat ska använda parenteser (som i finansiella program),
- att antalet siffror efter decimalavgränsaren är 5,
- att nollor ska läggas till efter talet för att uppnå fem decimalplatser,
- att decimalavgränsaren ska vara ett komma,
- att inga grupperingsavgränsare ska användas.

Obs! *När någon av dessa egenskaper ändras kommer talformatet inte längre motsvara det format som föredras för det angivna språkområdet. Använd dessa egenskaper endast när språkområdeskontrollen inte är viktig; när du behöver ha detaljerad kontroll över en enskild aspekt av formatet, till exempel antalet efterföljande nollor, eller när användaren begärde ändringen direkt, till exempel genom Kontrollpanelen i Windows.*

Tolka strängar som innehåller numeriska värden

Klassen NumberFormatter kan också extrahera numeriska värden från strängar som följer språkområdesspecifika formateringskrav. Metoden NumberFormatter.parseNumber() extraherar ett numeriskt värde från en sträng. Till exempel:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "-1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // -1234567.89
trace("Status:" + nf.lastOperationStatus); // noError
```

Metoden parseNumber() hanterar strängar som endast innehåller siffror och talformaterande tecken, till exempel minustecken och avgränsare. Om strängen innehåller andra tecken kommer en felkod att anges:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "The value is 1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // NaN
trace("Status:" + nf.lastOperationStatus); // parseError
```

Använd metoden NumberFormatter.parse() för att extrahera tal från strängar som innehåller andra alfabetiska tecken:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String = "The value is 123,456,7.890";
var parseResult:NumberParseResult = nf.parse(inputNumberString);
trace("Value:" + parseResult.value); // 1234567.89
trace("startIndex: " + parseResult.startIndex); // 14
trace("Status:" + nf.lastOperationStatus); // noError
```

Metoden parse() returnerar ett NumberParseResult-objekt som innehåller det tolkade numeriska värdet i egenskapen för värde. Egenskapen startIndex indikerar index för det första numeriska tecknet som hittades. Du kan använda egenskaperna startIndex och endIndex om du vill extrahera de delar av strängen som finns före och efter siffrorna.

Formatera valutavärden

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Visningsformaten för valutavärden varierar lika mycket som talformat. Här visas till exempel hur värdet 123456,78 $ formateras för olika språkområden:

Språk	Talformat
en-US (engelska, USA)	$123,456.78
de-DE (tyska, Tyskland)	123.456,78 $
en-IN (engelska, Indien)	$ 1,23,456.78

Valutaformatering styrs av samma faktorer som för talformatering, samt några faktorer till:

- ISO-kod för valuta. ISO 4217-valutakoden, tre bokstäver, för det språk som faktiskt används, till exempel USD eller EUR.
- Valutasymbol. Valutasymbolen eller strängen för det språk som faktiskt används, till exempel $ eller €.
- Negativt valutaformat. Definierar placeringen av valutasymbolen och den negativa symbolen eller parentesen i förhållande till den numeriska delen av valutabeloppet.
- Positivt valutaformat. Definierar placeringen av valutasymbolen i förhållande till den numeriska delen av valutabeloppet.

Använda klassen CurrencyFormatter

Klassen CurrencyFormatter formaterar numeriska värden till strängar som innehåller valutasträngar och formaterade tal, i enlighet med konventionerna för ett visst språkområde.

När du instansierar ett nytt CurrencyFormatter-objekt, anges valutan till standardvalutan för språkområdet.

I följande exempel visas att ett CurrencyFormatter-objekt som skapats med det tyska språkområdet utgår från att valutabelopp ska vara i euro:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "de-DE" );
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

Det är oftast bäst att inte förlita sig på standardvalutan för ett språkområde. Om användarens standardspråkområde inte stöds tilldelar klassen CurrencyFormatter ett alternativt språkområde. Det alternativa språkområdet kanske har en annan standardvaluta. Dessutom är det ofta viktigt att valutaformat ser korrekta ut för användaren, även om beloppen inte anges i användarens lokala valuta. En kanadensisk användare kanske vill se ett tyskt företags priser i euro, men formaterade enligt det kanadensiska formatet.

Metoden CurrencyFormatter.setCurrency() anger exakt vilken valutasträng och valutasymbol som ska användas.

I följande exempel visas valutabelopp i euro för användare i den fransktalande delen av Kanada:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "fr-CA" );
cf.setCurrency("EUR", "€");
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

Metoden setCurrency() kan också användas för större tydlighet genom användning av entydiga valutasymboler. Till exempel:

```
cf.setCurrency("USD","US$");
```

Som standard visar metoden format() en ISO 4217-valutakod som består av tre bokstäver, istället för en valutasymbol. ISO 4217-koder är entydiga och kräver ingen lokalisering. Men många användare föredrar valutasymboler istället för ISO-koder.

Klassen CurrencyFormatter kan hjälpa dig att avgöra vilken formaterad valutasträng som ska användas för en valutasymbol, till exempel ett dollartecken eller eurotecken, eller en ISO-valutasträng med tre bokstäver, till exempel USD eller EUR. Ett belopp i kanadensiska dollar kan visas som $200 för en användare i Kanada. För en användare i USA kommer det emellertid att visas som CAD 200. Använd metoden formattingWithCurrencySymbolIsSafe() för att avgöra om beloppets valutasymbol blir oklar eller felaktig beroende på användarens språkområdesinställningar.

I exemplet nedan formateras ett värde i euro till ett format för språkområdet en-US. Beroende på användarens språkområde används för utdatasträngen antingen ISO-valutakoden eller valutasymbolen.

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-CA");

if (cf.formattingWithCurrencySymbolIsSafe("USD"))
{
    trace(cf.format(1234567.89, true)); // $1,234,567.89
}
else
{
    cf.setCurrency("USD", "$");
    trace(cf.format(1234567.89)); // USD1,234,567.89
}
```

Tolka strängar som innehåller valutavärden

Klassen CurrencyFormatter kan också extrahera ett valutabelopp och en valutasträng från en indatasträng som följer en språkområdesspecifik formatering. Metoden CurrencyFormatter.parse() lagrar det tolkade beloppet och valutasträngen i ett CurrencyParseResult-objekt, enligt exemplet nedan:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "(GBP 123,56,7.890)";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("parsed amount: " + parseResult.value); // -1234567.89
trace("currencyString: " + parseResult.currencyString ); // GBP
```

Delen med valutasträngen i indatasträngen kan innehålla en valutasymbol, en ISO-valutakod och andra tecken. Placeringen av valutasträngen, indikatorn för negativa tal samt det numeriska värdet, överensstämmer med formaten som anges av egenskaperna negativeCurrencyFormat och positiveCurrencyFormat. Till exempel:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "Total $-123,56,7.890";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // parseError
trace("parsed amount: " + parseResult.value); // NaN
trace("currencyString: " + parseResult.currencyString ); //
cf.negativeCurrencyFormat = 2;
parseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // noError
trace("parsed amount: " + parseResult.value); // -123567.89
trace("currencyString: " + parseResult.currencyString ); // Total $
```

I det här exemplet innehåller indatasträngen en valutasträng som följs av ett minustecken och ett tal. Men standardvärdet för negativeCurrencyFormat för språkområdet en-US är att indikatorn för negativa tal ska anges först. Därför genererar metoden parse() ett fel och det tolkade värdet blir ”NaN”.

När negativeCurrencyFormat anges till 2, vilket innebär att valutasträngen kommer först lyckas parse()-metoden.

Formatera datum och tid

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Visningsformatet för värden för datum och tid varierar också mycket mellan olika regioner. I exemplet nedan visas hur värdet för den andra januari 1962, klockan 13:01, visas i kortformat i olika språkområden:

Språk	Datum- och tidsformat
en-US (engelska, USA)	1/2/62 1:01pm
fr-FR (franska, Frankrike)	2/1/62 13:01
ja-JP (japanska, Japan)	1962/2/1 13:01

Använda klassen DateTimeFormatter

Klassen DateTimeFormatter formaterar datumvärden till strängar för datum och tid i enlighet med konventionerna för ett visst språkområde.

Formateringen följer en mönstersträng som innehåller bokstavskombinationer som ersätts med värden för datum och tid. I mönstret "yyyy/MM" ersätts tecknen "yyyy" med ett fyrsiffrigt årtal, följt av tecknet "/", och ett tvåsiffrigt månadstal.

Mönstersträngen kan anges explicit med metoden setDateTimePattern(). Det är emellertid bäst att mönstret ställs in automatiskt enligt användarens inställningar för språkområde och operativsystem. Detta resulterar i att inställningarna görs till rätt kulturområde.

DateTimeFormatter kan visa datum och tid i tre olika standardformat: Långt format (LONG), medellångt format (MEDIUM) och kort format (SHORT), och det går även att använda ett anpassat mönsterformat (CUSTOM). Ett format kan användas för datum och ett annat för tid. De faktiska mönstren som används för varje format kan variera något mellan olika operativsystem.

Du kan ange formatet när du skapar ett DateTimeFormatter-objekt. Om inga formatparametrar är angivna kommer de som standard att anges till DateTimeStyle.LONG. Du kan ändra formaten senare med metoden setDateTimeStyles(), enligt exemplet nedan:

```
var date:Date = new Date(2009, 2, 27, 13, 1);
var dtf:DateTimeFormatter = new DateTimeFormatter("en-US",
    DateTimeStyle.LONG, DateTimeStyle.LONG);

var longDate:String = dtf.format(date);
trace(longDate); // March 27, 2009 1:01:00 PM

dtf.setDateTimeStyles(DateTimeStyle.SHORT, DateTimeStyle.SHORT);
var shortDate:String = dtf.format(date);
trace(shortDate); // 3/27/09 1:01 PM
```

Lokalisera namn på månader och dagar

Många program använder listor som innehåller namn på månader och veckodagar i kalendervisningar eller listrutor.

Du kan hämta en lokaliserad lista med namn på månader med metoden DateTimeFormatter.getMonthNames(). Beroende på olika operativsystem finns fullständiga och förkortade former tillgängliga. Skicka värdet DateTimeNameStyle.FULL om du vill hämta fullständiga namn på månader. Skicka värdena DateTimeNameStyle.LONG_ABBREVIATION eller DateTimeNameStyle.SHORT_ABBREVIATION om du vill hämta kortare versioner.

För vissa språk ändras namnet på månaden (till genitivform) när det står bredvid värdet för dag i ett datumformat. Om du planerar att använda namnen på månaderna fristående, ska du skicka värdet DateTimeNameContext.STANDALONE till metoden getMonthNames(). Om du emellertid använder månadsnamnen i formaterade datum, skickar du värdet DateTimeNameContext.FORMAT.

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var months:Vector.<String> = dtf.getMonthNames(DateTimeNameStyle.FULL,
        DateTimeNameContext.STANDALONE);
trace(months[0]); // janvier
months = dtf.getMonthNames(DateTimeNameStyle.SHORT_ABBREVIATION,
         DateTimeNameContext.STANDALONE);
trace(months[0]); // janv.
```

Metoden DateTimeFormatter.getWeekdayNames() tillhandahåller en lokaliserad lista av namn på veckodagar. Metoden getWeekdayNames() accepterar samma parametrar för nameStyle och context som metoden getMonthNames().

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var weekdays:Vector.<String> = dtf.getWeekdayNames(DateTimeNameStyle.FULL,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dimanche
weekdays = dtf.getWeekdayNames(DateTimeNameStyle.LONG_ABBREVIATION,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dim.
```

Dessutom returnerar metoden getFirstWeekday() indexvärdet för den dag som vanligtvis markerar början på veckan i det valda språkområdet.

Sortera och jämföra strängar

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Sortering innebär att ordna saker i rätt ordning. Sorteringsregler kan variera mycket mellan olika språkområden. Reglerna varierar också om du sorterar en lista eller matchar liknande objekt, som exempelvis i en textsökningsalgoritm.

Vid sortering är ofta små skillnader viktiga, som versala eller gemena bokstäver eller diakritiska tecken som accenter. Bokstaven "ö" (ett "o" med ett trema) tolkas ofta som bokstaven "o" i franska och engelska. Samma bokstav kommer emellertid i svenskan efter bokstaven "z". I franska och vissa andra språk avgör dessutom accenten i slutet av ett ord vilken ordning ordet ska ha i en sorterad lista.

För sökning är det ofta bäst att ignorera skillnader i versaler och gemener eller i diakritiska tecken för att kunna hitta fler relevanta träffar. En sökning efter teckenkombinationen ”cote” i ett franskt dokument ger till exempel träffar för ”cote”, ”côte” och ”coté”.

Använda klassen Collator

De huvudsakliga metoderna för klassen Collator är metoden compare(), som huvudsakligen används för sortering, och metoden equals(), som används för att matcha värden.

I följande exempel visas de olika beteendena för metoderna compare() och equals().

```
var words:Array = new  Array("coté", "côte");

var sorter:Collator = new Collator("fr-FR", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // côte,coté

var matcher:Collator = new Collator("fr-FR", CollatorMode.MATCHING);
if (matcher.equals(words[0], words[1]))
{
    trace(words[0] + " = " + words[1]); // côte = coté
}
```

I exemplet skapas först ett Collator-objekt i sorteringsläget SORTING för språkområdet franska-Frankrike. Sedan sorteras två ord som endast skiljer sig genom diakritiska tecken. Detta visar att SORTING-jämförelsen gör skillnad mellan tecken med eller utan accent.

Sorteringen sker genom att en referens skickas till metoden sort() för Collator-objektet som en parameter till metoden Array.sort(). Den här tekniken är ett av de mest effektiva sätten att kontrollera sorteringsordningen med ett Collator-objekt.

I exemplet skapas sedan ett Collator-objekt i matchningsläget MATCHING. När de båda orden jämförs av Collator-objektet behandlas de som likvärdiga. Detta visar att MATCHING-jämförelsen inte gör någon skillnad mellan tecken med eller utan accent.

Anpassa beteendet för klassen Collator

Jämförelseregler för klassen Collator fås som standard från operativsystemet baserat på språkområde och på användarens systeminställningar. Du kan anpassa beteendet för metoderna compare() och equals() genom att explicit ange olika egenskaper. I följande tabell visas egenskaperna och vilken effekt de har på jämförelsen:

Collator-egenskap	Effekt
numericComparison	Definierar om numeriska tecken ska behandlas som siffror eller som text.
ignoreCase	Kontrollerar om versal- och gemenskillnader ignoreras.
ignoreCharacterWidth	Definierar om former med full bredd och halv bredd för vissa kinesiska och japanska tecken ska behandlas som likvärdiga.
ignoreDiacritics	Definierar om strängar som innehåller samma grundtecken men med olika accenter eller andra diakritiska tecken ska behandlas som likvärdiga.
ignoreKanaType	Definierar om strängar som endast skiljer sig åt genom typen av kana-tecken ska behandlas som likvärdiga.
ignoreSymbols	Definierar om symboltecken som blanksteg, valutasymboler, matematiska symboler och andra symboler ska ignoreras eller inte.

I följande kodexempel visas att sorteringsordningen för en lista med franska ord ändras om egenskapen ignoreDiacritics anges till true:

```
var words:Array = new  Array("COTE", "coté", "côte", "Coté","cote");
var sorter:Collator = new Collator("fr-CA", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // cote,COTE,côte,coté,Coté

sorter.ignoreDiacritics = true;
words.sort(sorter.compare);
trace(words); // côte,coté,cote,Coté,COTE
```

Skiftlägeskonvertering

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Olika språk har också olika regler för hur bokstäver konverteras mellan versaler (majuskler) och gemener (minuskler).

Till exempel är ”i” den gemena formen av den versala bokstaven ”I” i de flesta språk som använder det latinska alfabetet. Men i vissa språk (som turkiska och azeriska) finns även tecknet ”ı”, utan pricken över i:et. Detta innebär att i dessa språk kommer ett pricklöst "ı" att omformas till ett versalt "I". Gemena "i" omformas till versala "İ" med en punkt.

Klassen StringTools har metoder med språkspecifika regler för sådana konverteringar.

Använda klassen StringTools

Klassen StringTools har två metoder för skiftlägeskonvertering: toLowerCase() och toUpperCase(). StringTools-objekt skapas genom att anropa konstruktorn med ett språkområdes-ID. Klassen StringTools hämtar reglerna för skiftlägeskonvertering för detta språkområde (eller för ett reservspråkområde) från operativsystemet. Det är inte möjligt att anpassa algoritmen för skiftlägeskonvertering ytterligare.

I följande exempel används metoderna toUpperCase() och toLowerCase() för att konvertera ett tyskt ord som innehåller bokstaven ”ß” (så kallat dubbel-s).

```
var phrase:String = "Schloß Neuschwanstein";
var converter:StringTools = new StringTools("de-DE");

var upperPhrase:String = converter.toUpperCase(phrase);
trace(upperPhrase); // SCHLOSS NEUSCHWANSTEIN

var lowerPhrase:String = converter.toLowerCase(upperPhrase);
trace(lowerPhrase);// schloss neuschwanstein
```

Metoden toUpperCase() omformar den gemena bokstaven "ß" till de versala bokstäverna "SS". Den här omformningen fungerar bara åt ett håll. När bokstäverna "SS" konverteras tillbaka till gemena bokstäver blir resultatet "ss", och inte "ß".

Exempel: Internationalisering av ett börskursprogram

Flash Player 10.1 och senare, Adobe AIR 2.0 och senare

Programmet Global Stock Ticker hämtar och visar fiktiva data om börskurserna för tre olika börsmarknader: USA, Japan och Europa. Dessa data formateras i programmet efter konventionerna som gäller för de olika språkområdena.

I det här exemplet beskrivs följande funktioner i flash.globalization-paketet:

- Språkområdeskänslig talformatering
- Språkområdeskänslig valutaformatering
- Ange ISO-valutakod och valutasymboler
- Språkområdeskänslig datumformatering
- Hämta och visa korrekta namn och förkortningar för månader

Programfilerna för det här exemplet finns på www.adobe.com/go/learn_programmingAS3samples_flash_se. Filerna för programmet Global Stock Ticker finns i mappen Samples/GlobalStockTicker. Programmet består av följande filer:

Fil	Beskrivning
GlobalStockTicker.mxml eller GlobalStockTicker.fla	Användargränssnittet för programmet för Flex (MXML) eller Flash (FLA).
styles.css	Format för programmets användargränssnitt (endast Flex).
com/example/programmingas3/stockticker/flex/FinGraph.mxml	En MXML-komponent som visar ett diagram över fiktiva börskursdata (endast Flex).
com/example/programmingas3/stockticker/flash/GlobalStockTicker.as	Dokumentklass som innehåller användargränssnittslogiken för programmet (endast Flash).
comp/example/programmingas3/stockticker/flash/RightAlignedColumn.as	En anpassad cellrenderare för Flash-komponenten DataGrid (endast Flash).

Fil	Beskrivning
com/example/programmingas3/stockticker/FinancialGraph.as	En ActionScript-klass som ritar ett diagram över fiktiva börskursdata.
com/example/programmingas3/stockticker/Localizer.as	En ActionScript-klass som hanterar språkområdet och valutan samt den lokaliserade formateringen av tal, valutabelopp och datum.
com/example/programmingas3/stockticker/StockDataModel.as	En ActionScript-klass som innehåller alla exempeldata för exemplet Global Stock Ticker.

Förstå användargränssnittet och exempeldata

Elementen i programmets huvudanvändargränssnitt är:

- en kombinationsruta för val av språkområde
- en kombinationsruta för val av marknad
- en DataGrid som visar data för sex företag i varje marknad
- ett diagram som visar fiktiva historiska data för det valda företagets börskurs

Alla exempeldata om språkområden, marknader och företagens börskurser lagras i klassen StockDataModel. Om programmet skulle vara på riktigt skulle data hämtas från en server och sedan lagras i en klass som exempelvis StockDataModel. I det här exemplet är alla data hårdkodade i klassen StockDataModel.

Obs! *De data som visas i diagrammet matchar inte nödvändigtvis de data som visas i kontrollen DataGrid. Diagrammet ritas slumpmässigt om varje gång ett annat företag väljs. Detta är endast ett exempel.*

Ange språkområde

Efter en inledande konfiguration anropar programmet metoden Localizer.setLocale() för att skapa formateringsobjekt för standardspråkområdet. Metoden setLocale() anropas också varje gång användaren väljer ett nytt värde i kombinationsrutan för språkområde.

```
public function setLocale(newLocale:String):void
{
    locale  = new LocaleID(newLocale);

    nf = new NumberFormatter(locale.name);
    traceError(nf.lastOperationStatus, "NumberFormatter", nf.actualLocaleIDName);

    cf = new CurrencyFormatter(locale.name);
    traceError(cf.lastOperationStatus, "CurrencyFormatter", cf.actualLocaleIDName);
    symbolIsSafe = cf.formattingWithCurrencySymbolIsSafe(currentCurrency);
    cf.setCurrency(currentCurrency, currentSymbol);
    cf.fractionalDigits = currentFraction;

    df = new DateTimeFormatter(locale.name, DateTimeStyle.LONG, DateTimeStyle.SHORT);
    traceError(df.lastOperationStatus, "DateTimeFormatter", df.actualLocaleIDName);
    monthNames = df.getMonthNames(DateTimeNameStyle.LONG_ABBREVIATION);
}

public function traceError(status:String, serviceName:String, localeID:String) :void
{
```

```
    if(status != LastOperationStatus.NO_ERROR)
    {
        if(status == LastOperationStatus.USING_FALLBACK_WARNING)
        {
            trace("Warning - Fallback locale ID used by "
                    + serviceName + ": " + localeID);
        }
        else if (status == LastOperationStatus.UNSUPPORTED_ERROR)
        {
            trace("Error in " + serviceName + ": " + status);
            //abort application
            throw(new Error("Fatal error", 0));
        }
        else
        {
            trace("Error in " + serviceName + ": " + status);
        }
    }
    else
    {
        trace(serviceName + " created for locale ID: " + localeID);
    }
}
```

Först skapar metoden setLocale() ett LocaleID-objekt. Detta objekt gör det lättare att få information om det faktiska språkområdet senare om det behövs.

Sedan skapas nya NumberFormatter-, CurrencyFormatter- och DateTimeFormatter-objekt för språkområdet. Sedan varje formatter-objekt har skapats anropas metoden traceError(). Denna metod visar fel- och varningsmeddelanden i konsolen om det finns några problem med det aktuella språkområdet. (I ett verkligt program skulle dessa fel åtgärdas i stället för att endast spåras).

När CurrencyFormatter-objektet har skapats anger metoden setLocale() formateringens ISO-valutakod, valutasymbol och fractionalDigits-egenskaper för tidigare angivna värden. (Dessa värden anges varje gång användaren väljer en annan marknad i kombinationsrutan för marknader).

När DateTimeFormatter-objektet har skapats hämtar metoden setLocale() även en matris av lokaliserade förkortningar på månader.

Formatera data

Formaterade börskursdata visas i kontrollen DataGrid. Varje DataGrid-kolumn anropar en etikettfunktion som formaterar kolumnvärdet med lämpligt formateringsobjekt.

I Flash-versionen anges DataGrid-kolumnerna med följande kod:

```
var col1:DataGridColumn = new DataGridColumn("ticker");
col1.headerText = "Company";
col1.sortOptions = Array.NUMERIC;
col1.width = 200;

var col2:DataGridColumn = new DataGridColumn("volume");
col2.headerText = "Volume";
col2.width = 120;
col2.cellRenderer = RightAlignedCell;
col2.labelFunction = displayVolume;

var col3:DataGridColumn = new DataGridColumn("price");
col3.headerText = "Price";
col3.width = 70;
col3.cellRenderer = RightAlignedCell;
col3.labelFunction = displayPrice;

var col4:DataGridColumn = new DataGridColumn("change");
col4.headerText = "Change";
col4.width = 120;
col4.cellRenderer = RightAlignedCell;
col4.labelFunction = displayPercent;
```

I Flex-versionen av exemplet deklareras DataGrid i MXML. Liknande etikettfunktioner definieras också för varje kolumn.

Egenskaperna för labelFunction avser följande funktioner, som anropar formateringsmetoder för klassen Localizer:

```
private function displayVolume(item:Object):String
{
    return localizer.formatNumber(item.volume, 0);
}

private function displayPercent(item:Object):String
{
    return localizer.formatPercent(item.change )  ;
}

  private function displayPrice(item:Object):String
{
    return localizer.formatCurrency(item.price);
}
```

Localizer-metoderna anger sedan och anropar lämplig formatering:

```
public function formatNumber(value:Number, fractionalDigits:int = 2):String
{
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value);
}

public function formatPercent(value:Number, fractionalDigits:int = 2):String
{
    // HACK WARNING: The position of the percent sign, and whether a space belongs
    // between it and the number, are locale-sensitive decisions. For example,
    // in Turkish the positive format is %12 and the negative format is -%12.
    // Like most operating systems, flash.globalization classes do not currently
    // provide an API for percentage formatting.
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value) + "%";
}

public function formatCurrency(value:Number):String
{
    return cf.format(value, symbolIsSafe);
}

public function formatDate(dateValue:Date):String
{
    return df.format(dateValue);
}
|
```

Kapitel 57: Lokalisera program

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Lokalisering är en process där du inkluderar resurser som kan hantera flera språkområden. Ett språkområde är en kombination av en språkkod och en landskod. Till exempel avser språkområdet en_US den variant av engelska som talas i USA och fr_FR den franska som talas i Frankrike. Om du vill lokalisera ett program för dessa språkområden måste du ange två uppsättningar resurser: en för språkområdet en_US och en för fr_FR.

Olika språkområden kan ha samma språk. Till exempel är en_US och en_GB (Storbritannien) olika språkområden. I det här fallet används engelska för båda språkområden men landskoden visar att det är olika språkområden och att de därför kanske använder olika resurser. I ett program med språkområdet en_US kan till exempel ordet färg stavas som ”color”, medan det stavas ”colour” med språkområdet en_GB. Valutaenheter visas i dollar eller pund beroende på språkområde, och datum- och tidsformaten kan också vara olika.

Du kan ange en uppsättning resurser för ett språk utan att ange en landskod. Du kan till exempel ange en-resurser för engelska och ange ytterligare resurser för språkområdet en_US, som endast ska gälla för amerikansk engelska.

Lokalisering handlar om mer än att bara översätta strängar som används i programmet. Det kan även omfatta olika resurstyper, t.ex. ljudfiler, bilder och videoklipp.

Välja en språkinställning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill se vilken språkinställning som används i innehållet eller programmet kan du göra något av följande:

- flash.globalization package – Använd språkområdeskänsliga klasser i flash.globalization-paketet för att hämta standardspråkområdet för användaren utifrån operativsystemet och användarinställningarna. Det här är det rekommenderade sättet för program som körs på Flash Player 10.1 eller senare eller AIR 2.0 eller senare. Se ”Bestämma språkområde” på sidan 947för mer information.
- Användaruppmaning – Du kan starta programmet med en viss standardspråkinställning och sedan be användaren att välja önskad språkinställning.
- `(endast AIR) Capabilities.languages` – Egenskapen `Capabilities.languages` listar en array med de språk som är tillgängliga, utifrån de språkinställningar användaren gjort via operativsystemet. Strängarna innehåller språktaggar (och skript och regionsinformation, om sådan finns) enligt RFC4646 (http://www.ietf.org/rfc/rfc4646.txt). I strängarna används bindestreck som avgränsare (t.ex. `"en-US"` och `"ja-JP"`). Den första posten i den returnerade arrayen har samma primära språk-ID som language-egenskapen. Om till exempel `languages[0]` är inställd på `"en-US"` ställs `language`-egenskapen in på `"en"`. Men om language-egenskapen ställs in på `"xu"` (ett okänt språk) är det första elementet i `languages`-arrayen annorlunda.
- `Capabilities.language` – `Capabilities.language`-egenskapen tillhandahåller språkkoden för användargränssnittet i operativsystemet. Den här egenskapen är dock begränsad till 20 kända språk. I engelska system returnerar den här egenskapen bara språkkoden, inte landskoden. På grund av det här är det bättre att använda det första elementet i `Capabilities.languages`-arrayen.

Lokalisera Flex-innehåll

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Adobe Flex finns ett ramverk för lokalisering av Flex-innehåll. Ramverket omfattar klasserna Locale, ResourceBundle och ResourceManagerImpl, samt gränssnitten IResourceBundle och IResourceManagerImpl.

Ett Flex-lokaliseringsbibliotek med hjälpklasser för att sortera programspråkområden finns tillgängligt på Google Code (http://code.google.com/p/as3localelib/).

Fler hjälpavsnitt

http://code.google.com/p/as3localelib/

Lokalisera Flash-innehåll

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Adobe Flash Professional omfattar en Locale-klass i ActionScript 3.0-komponenterna. Med hjälp av klassen Locale kan du styra hur en SWF-fil visar text på flera språk. På strängpanelen i Flash kan du använda sträng-ID i stället för stränglitteraler i dynamiska textfält. Det innebär att du kan skapa en SWF-fil som visar text som läses in från språkspecifika XML-filer. Information om hur du använder klassen Locale finns i Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen.

Lokalisera AIR-program

Adobe AIR 1.0 och senare

I AIR SDK finns ett HTML-lokaliseringsramverk (finns i filen AIRLocalizer.js). Detta ramverk innehåller API:er som kommer väl till pass när du arbetar med flera språkområden i ett HTML-baserat program. Ett ActionScript-bibliotek för att sortera språkområden finns på http://code.google.com/p/as3localelib/.

Fler hjälpavsnitt

http://code.google.com/p/as3localelib/

Lokalisera datum, tider och valutor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Visningen av datum, tider och valutor i ett program varierar väldigt mycket för varje språkinställning. Den amerikanska standarden för att visa datum är till exempel månad/dag/år medan den europeiska standarden är dag/månad/år.

Du kan skriva kod för att formatera datum, tider och valutor. Följande kod konverterar till exempel ett Date-objekt till formatet månad/dag/år eller formatet dag/månad/år. Om variabeln `locale` (representerar språkinställningen) är angiven som `"en_US"`, returnerar funktionen formatet månad/dag/år. Exemplet konverterar ett Date-objekt till formatet dag/månad/år för alla andra språkinställningar:

```
function convertDate(date)
{
    if (locale == "en_US")
    {
        return (date.getMonth() + 1) + "/" + date.getDate() + "/" + date.getFullYear();
    }
    else
    {
        return date.getDate() + "/" + (date.getMonth() + 1) + "/" + date.getFullYear();
    }
}
```

ADOBE FLEX

I Flex-ramverket finns kontroller för formatering av datum, tider och valutor. Dessa omfattar kontrollerna DateFormatter och CurrencyFormatter.

- mx:DateFormatter
- mx:CurrencyFormatter

Kapitel 58: Om HTML-miljön

Adobe AIR 1.0 och senare

Adobe® AIR® använder WebKit (*www.webkit.org*), som också finns i webbläsaren Safari, för att tolka, utforma och återge HTML- och JavaScript-innehåll. Det är valfritt att använda AIR-API:er i HTML-innehåll. Du kan programmera innehållet i ett HTMLLoader-objekt eller HTML-fönster helt med HTML och JavaScript. De flesta befintliga HTML-sidor och -program bör kunna köras med några få ändringar (under förutsättning att de använder HTML-, CSS-, DOM- och JavaScript-funktioner som är kompatibla med WebKit).

Viktigt! Nya versioner av Adobe AIR kan inkludera uppdaterade versioner av WebKit. En WebKit-uppdatering i en ny version av AIR *kan* orsaka oväntade förändringar i ett distribuerat AIR-program. De här ändringarna kan påverka beteendet för eller utseendet på HTML-innehåll i ett program. Förbättringar eller korrigeringar i WebKit-återgivningen kan till exempel påverka elementens layout i programmets användargränssnitt. Därför bör du tillhandahålla en uppdateringsfunktion i ditt program. Om du skulle behöva uppdatera programmet på grund av ändringar i den WebKit-version som ingår i AIR kan uppdateringsfunktionen i AIR uppmana användaren att installera den nya versionen av ditt program.

Av följande tabell framgår vilken version av webbläsaren Safari som använder den WebKit-version som motsvarar den som används i AIR:

AIR-version	Safari-version
1.0	2.04
1.1	3.04
1.5	4.0 Beta
2.0	4.03
2.5	4.03
2.6	4.03
2.7	4.03
3	5.0.3

Du kan alltid ta reda på vilken version av WebKit som är installerad genom att titta i strängen för standardanvändaragent som returneras av HTMLLoader-objekt:

```
var htmlLoader:HTMLLoader = new HTMLLoader();
trace( htmlLoader.userAgent );
```

Kom ihåg att den WebKit-version som används i AIR inte är identisk med versionen med öppen källkod. Vissa funktioner stöds inte i AIR och AIR-versionen kan även innehålla säkerhets- och buggfixar som inte ännu finns tillgängliga för den motsvarande WebKit-versionen. Se ”WebKit-funktioner som inte stöds i AIR” på sidan 980.

Eftersom AIR-program körs direkt på skrivbordet, med fullständig åtkomst till filsystemet, är säkerhetsmodellen för HTML-innehållet mer tvingande än säkerhetsmodellen för en vanlig webbläsare. I AIR placeras bara innehåll som läses in från programinstallationskatalogen i *sandlådan application*. Sandlådan application har den högsta behörighetsnivån och ger tillgång till AIR-API:er. Annat innehåll placeras i isolerade sandlådor utifrån var innehåller kommer från. Filer som läses in från filsystemet placeras i en lokal sandlåda. Filer som läses in från nätverket med http- eller https- protokoll placeras i en sandlåda utifrån domänen på fjärrservern. Innehåll i sandlådor som inte är programsandlådor har inte tillgång till några AIR-API:er och körs ungefär som i en vanlig webbläsare.

HTML-innehåll i AIR visar inte SWF- eller PDF-innehåll om alfa-, skalnings- eller genomskinlighetsinställningar tillämpas. Mer information finns i ”Att tänka på vid inläsning av SWF- eller PDF-innehåll i en HTML-sida” på sidan 1010 och ”Fönstergenomskinlighet” på sidan 897.

Fler hjälpavsnitt

Webkit DOM Reference

Safari HTML Reference

Safari CSS Reference

www.webkit.org

Översikt över HTML-miljön

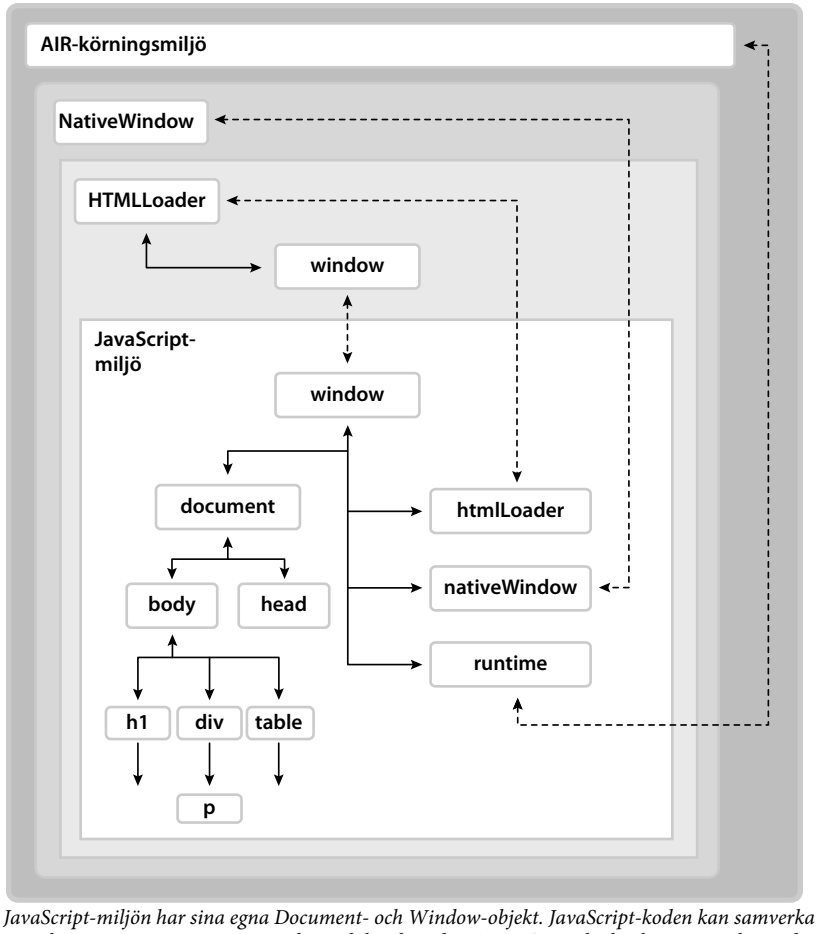
Adobe AIR 1.0 och senare

Adobe AIR innehåller en komplett webbläsarliknande JavaScript-miljö med ett HTML-återgivningstillägg, en dokumentobjektmodell och en JavaScript-tolk. JavaScript-miljön representeras av AIR-klassen HTMLLoader. I HTML-fönster innehåller ett HTMLLoader-objekt allt HTML-innehåll, och det finns i sin tur inuti ett NativeWindow-objekt. I SWF-innehåll kan klassen HTMLLoader, som utökar klassen Sprite, läggas till i visningslistan för en scen som alla andra visningsobjekt. Klassens egenskaper i Adobe® ActionScript® 3.0 beskrivs i ”Skripta AIR HTML-behållaren” på sidan 1008 och i *Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen*. I Flex-ramverket placeras AIR-klassen HTMLLoader i en mx:HTML-komponent. mx:HTML-komponenten utökar klassen UIComponent så att den kan användas direkt med andra Flex-behållare. JavaScript-miljön i mx:HTML-komponenten är annars identisk.

Om JavaScript-miljön och dess förhållande till AIR-värden

Adobe AIR 1.0 och senare

I bilden nedan visas sambandet mellan JavaScript-miljön och AIR-körningsmiljön. Även om bara ett enda inbyggt fönster visas, kan ett AIR-program innehålla flera fönster. (Och ett enda fönster kan innehålla flera HTMLLoader-objekt.)



JavaScript-miljön har sina egna Document- och Window-objekt. JavaScript-koden kan samverka med AIR-körningsmiljön genom egenskaperna runtime, nativeWindow och htmlLoader. ActionScript-koden kan samverka med JavaScript-miljön genom egenskapen window för ett HTMLLoader-objekt, som är en referens till JavaScript-objektet Window. Både ActionScript- och JavaScript-objekt kan dessutom avlyssna händelser som skickas av både AIR- och JavaScript-objekt.

Egenskapen `runtime` ger tillgång till AIR API-klasser, så att du kan skapa nya AIR-objekt och få tillgång till klassmedlemmar (kallas även för statiska medlemmar). När du vill få tillgång till ett AIR-API lägger du till namnet på klassen, med paketet, i egenskapen `runtime`. Om du till exempel vill skapa ett File-objekt använder du programsatsen:

```
var file = new window.runtime.filesystem.File();
```

Obs! *AIR SDK innehåller en JavaScript-fil, `AIRAliases.js`, som definierar mer praktiska alias för de flesta vanliga AIR-klasser. När du importerar den här filen kan du använda den kortare formen air.Class i stället för window.runtime.package.Class. Du kan till exempel skapa File-objektet med `new air.File()`.*

Objektet NativeWindow innehåller egenskaper som styr skrivbordsfönstret. Från en HTML-sida får du tillgång till det innehållande NativeWindow-objektet med egenskapen `window.nativeWindow`.

HTMLLoader-objektet innehåller egenskaper, metoder och händelser som styr hur innehållet läses in och återges. Från en HTML-sida får du tillgång till det överordnade HTMLLoader-objektet med egenskapen `window.htmlLoader`.

Viktigt! *Det är bara sidor som installeras som en del av programmet som har egenskaperna* `htmlLoader`, `nativeWindow` *eller* `runtime` *och bara när de läses in som ett dokument på den översta nivån. Dessa egenskaper läggs inte till när dokumentet läses in i en bildruta eller iframe. (Ett underordnat dokument kan få tillgång till dessa egenskaper i det överordnade dokumentet så länge det finns i samma säkerhetssandlåda. Ett dokument som till exempel läses in i en bildruta har åtkomst till egenskapen* `runtime` *för dess överordnade via* `parent.runtime`*.)*

Om säkerhet

Adobe AIR 1.0 och senare

AIR kör all kod i en säkerhetssandlåda baserat på ursprungsdomänen. Programinnehåll, som begränsas till innehåll som läses in från programinstallationskatalogen, placeras i sandlådan *application*. Det är bara HTML och JavaScript som körs i den här sandlådan som kan få åtkomst till körningsmiljön och AIR-API:erna. Samtidigt blockeras den mest dynamiska utvärderingen och körningen av JavaScript i sandlådan application efter att alla hanterare för sidhändelsen `load` har returnerats.

Du kan mappa en programsida till en sandlåda som inte är en application-sandlåda genom att läsa in sidan i bildrutan eller en iframe och ställa in AIR-specifika `sandboxRoot`- och `documentRoot`-attribut för bildrutan. Genom att ställa in värdet `sandboxRoot` på den faktiska fjärrdomänen, kan du aktivera sandlådeinnehåll till att korsskripta innehållet i den domänen. Att mappa sidor på det här sättet kan vara användbart när du läser in och skriptar fjärrinnehåll, till exempel i ett *kombinationsprogram*.

Ett annat sätt att tillåta programinnehåll och annat innehåll att korsskripta varandra, och det enda sättet att ge icke programbaserat innehåll åtkomst till AIR-API:er, är att skapa en *sandlådebrygga*. Med en brygga av typen *överordnad-till-underordnad* kan innehåll i en underordnad bildruta, iframe eller fönster få åtkomst till bestämda metoder och egenskaper som definieras i sandlådan application. Med en brygga av typen *underordnad-till-överordnad* kan programinnehåll få åtkomst till bestämda metoder och egenskaper som definieras i sandlådan för den underordnade. Du skapar sandlådebryggor genom att ställa in egenskaperna `parentSandboxBridge` och `childSandboxBridge` för fönsterobjektet. Mer information finns i ”HTML-säkerhet i Adobe AIR” på sidan 1085 och ”Bildrute- och iframe-element i HTML” på sidan 976.

Om plug-iner och inbäddade objekt

Adobe AIR 1.0 och senare

AIR har stöd för plug-inen Adobe® Acrobat®. Användarna måste ha Acrobat eller Adobe® Reader® 8.1 (eller senare) för att kunna visa PDF-innehåll. Objektet HTMLLoader tillhandahåller en egenskap som kontrollerar om användarens system kan visa PDF. SWF-filsinnehållet kan också visas i HTML-miljön, men denna funktion är inbyggd i AIR och använder inte någon extern plug-in.

Inga andra Webkit-plugin-program stöds i AIR.

Fler hjälpavsnitt

”HTML-säkerhet i Adobe AIR” på sidan 1085

”HTML-sandlådor” på sidan 969

”Bildrute- och iframe-element i HTML” på sidan 976

”JavaScript-objektet Window” på sidan 974

”Objektet XMLHttpRequest” på sidan 970

”Lägga till PDF-innehåll i AIR” på sidan 551

AIR och WebKit

Adobe AIR 1.0 och senare

Adobe AIR använder WebKit-motorn med öppen källkod, som också används i webbläsaren Safari. AIR tillför flera tillägg som tillåter åtkomst till körningsklasser och objekt samt för säkerhet. WebKit tillför också funktioner som inte finns i W3C-standarderna för HTML, CSS och JavaScript.

Här beskrivs bara AIR-tilläggen och de viktigaste WebKit-tilläggen. Mer information om HTML, CSS och JavaScript som inte är standard finns på www.webkit.org och developer.apple.com. Information om standarder finns på W3C-webbplatsen. Mozilla tillhandahåller också en användbar allmän referens om HTML, CSS och DOM (WebKit- och Mozilla-motorerna är förstås inte identiska).

JavaScript i AIR

Flash Player 9 och senare, Adobe AIR 1.0 och senare

AIR gör flera ändringar av det normala beteendet för vanliga JavaScript-objekt. Många av dessa ändringar gör det enklare att skriva säkra program i AIR. Samtidigt innebär dessa ändringar i beteendet att vissa vanliga JavaScript-kodningsmönster, och befintliga webbprogram som använder dessa mönster, kanske inte alltid kan köras som förväntat i AIR. Information om hur du rättar till dessa typer av problem finns i ”Undvika säkerhetsrelaterade JavaScript-fel” på sidan 987.

HTML-sandlådor

Adobe AIR 1.0 och senare

AIR placerar innehåll i isolerade sandlådor enligt innehållets ursprung. Sandlådereglerna följer principen för samma ursprung som används i de flesta webbläsare, samt reglerna för sandlådor som används i Adobe Flash Player. AIR tillhandahåller dessutom en ny sandlåda av typen *application* som innehåller och skyddar programinnehåll. Mer information om vilka typer av sandlådor som du kan träffa på när du utvecklar AIR-program finns i ”Säkerhetssandlådor” på sidan 1049.

Det är bara HTML och JavaScript som körs i sandlådan application som kan få åtkomst till körningsmiljön och AIR-API:erna. Samtidigt är dynamisk utvärdering och körning av JavaScript, i dess olika former, starkt begränsad inom sandlådan application av säkerhetsskäl. Dessa begränsningar tillämpas oavsett om programmet läser in information direkt från en server eller inte. (Även filinnehåll, inklistrade strängar och direkta användarinmatningar kan vara otillförlitliga.)

Ursprunget till innehållet på en sida avgör vilken sandlåda som ska användas. Det är bara innehåll som läses in från programkatalogen (installationskatalogen som refereras av URL-schemat `app:`) som placeras i sandlådan application. Innehåll som läses in från filsystemet placeras i sandlådan *lokal-med-filsystem* eller *lokal-tillförlitlig*, som också medger åtkomst och interaktion med innehåll i det lokala filsystemet, men inte fjärrinnehåll. Innehåll som läses in från nätverket placeras i en fjärrsandlåda som motsvarar ursprungsdomänen.

Om du vill tillåta programsidan att samverka fritt med innehåll i en fjärrsandlåda, kan du mappa sidan till samma domän som fjärrinnehållet. Om du till exempel skriver ett program som visar kartuppgifter från en Internettjänst, kan sidan i programmet som läser in och visar innehåll från tjänsten mappas till tjänstedomänen. Attributen för mappning av sidor till en fjärrsandlåda och domän är nya, och har lagts till i HTML-elementen bildruta och iframe.

Om du vill tillåta innehåll i sandlådor av annan typ än application att använda AIR-funktioner på ett säkert sätt, kan du ställa in en överordnad sandlådebrygga. Om du vill tillåta programinnehåll att säkert anropa metoder och få tillgång till egenskaper för innehåll i andra sandlådor, kan du ställa in en underordnad sandlådebrygga. Säkerhet innebär att fjärrinnehåll inte av misstag kan hämta referenser till objekt, egenskaper eller metoder som inte exponeras uttryckligen. Enbart enkla datatyper, funktioner och anonyma objekt kan skickas via bryggan. Du måste emellertid undvika att uttryckligen exponera eventuellt farliga funktioner. Om du till exempel exponerar ett gränssnitt som tillåter att fjärrinnehåll läser och skriver filer var som helst i användarens system, kanske du gör det möjligt för fjärrinnehållet att ställa till med skada.

JavaScript-funktionen eval()

Adobe AIR 1.0 och senare

Det går bara att använda funktionen `eval()` i sandlådan application en gång när en sida har lästs in. Vissa användningsområden tillåts inte så att JSON-formaterade data kan tolkas på ett säkert sätt, men alla utvärderingar som leder till körbara programsatser ger ett fel. I ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087 beskrivs hur funktionen `eval()` får användas.

Funktionskonstruktorer

Adobe AIR 1.0 och senare

I sandlådan application kan funktionskonstruktorer användas innan en sida har lästs in. När sidans alla `load`-händelsehanterare har slutförts kan nya funktioner inte skapas.

Läsa in externa skript

Adobe AIR 1.0 och senare

HTML-sidor i sandlådan application kan inte använda taggen `script` för att läsa in JavaScript-filer utanför programkatalogen. För att en sida i programmet ska kunna läsa in ett skript utanför programkatalogen, måste den mappas till en annan sandlåda än application.

Objektet XMLHttpRequest

Adobe AIR 1.0 och senare

AIR innehåller ett XMLHttpRequest-objekt (XHR) som program kan använda för att begära data. I följande exempel visas en enkel databegäran:

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

I motsats till en webbläsare tillåter AIR att innehåll som körs i sandlådan application begär data från valfri domän. Resultatet från en XHR som innehåller en JSON-sträng kan utvärderas till dataobjekt, såvida inte resultatet också innehåller körbar kod. Om det finns körbara programsatser i XHR-resultatet returneras ett fel och utvärderingsförsöket misslyckas.

Kodinjektioner från fjärrkällor förhindras genom att synkrona XHR:er returnerar ett tomt resultat om de görs innan en sida har laddats färdigt. Asynkrona XHR:er returneras alltid efter att sidan har laddats.

AIR blockerar som standard korsdomäns-XMLHttpRequests i andra sandlådor än application. Ett överordnat fönster i sandlådan application kan välja att tillåta korsdomänsförfrågningar i en underordnad bildruta som rymmer innehåll i en annan sandlåda än application genom att ställa in `allowCrossDomainXHR`, ett attribut som tillförs av AIR, på `true` i det innehållande bildrute- eller iframe-elementet:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

Obs! *När det passar kan AIR-klassen URLStream också användas för att hämta data.*

Om du skickar en XMLHttpRequest till en fjärrserver från en bildruta eller iframe som rymmer programinnehåll som har mappats till en fjärrsandlåda, ska du se till att mappnings-URL:en inte maskar serveradressen som används i XHR. Ta en titt på följande iframe-definition, som mappar programinnehåll till en fjärrsandlåda för domänen example.com:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

Eftersom attributet `sandboxRoot` mappar om rot-URL:en för adressen www.example.com, läses alla förfrågningar in från programkatalogen och inte från fjärrservern. Förfrågningar mappas om oavsett om de härleds från sidnavigering eller från en XMLHttpRequest.

Om du vill undvika att blockera dataförfrågningar till fjärrservern av misstag, ska du mappa `sandboxRoot` till en underkatalog hos fjärr-URL:en i stället för roten. Katalogen behöver inte finnas. Om du till exempel vill tillåta att förfrågningar till www.example.com läses in från fjärrservern i stället för programkatalogen, ändrar du föregående iframe till följande:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

I det här fallet läses bara innehåll i underkatalogen `air` in lokalt.

Mer information om mappning av sandlådor finns i ”Bildrute- och iframe-element i HTML” på sidan 976 och ”HTML-säkerhet i Adobe AIR” på sidan 1085.

Cookies

Adobe AIR 1.0 och senare

I AIR-program kan bara innehåll i fjärrsandlådor (innehåll som läses in från http- och https- källor) använda cookies (egenskapen `document.cookie`). Andra sätt att spara beständiga data finns i programmets sandlåda, bland annat via klasserna EncryptedLocalStore, SharedObject och FileStream.

Objektet Clipboard

Adobe AIR 1.0 och senare

API:et WebKit Clipboard drivs av följande händelser: `copy`, `cut` och `paste`. Händelseobjektet som skickas i dessa händelser ger åtkomst till Urklipp via egenskapen `clipboardData`. Använd följande metoder för objektet `clipboardData` när du vill läsa eller skriva urklippsdata:

Metod	Beskrivning
clearData(mimeType)	Rensar urklippsdata. Ställ in parametern `mimeType` på MIME-typen för de data som ska rensas.
getData(mimeType)	Hämta urklippsdata. Den här metoden kan bara anropas i en hanterare för händelsen `paste`. Ställ in parametern `mimeType` på MIME-typen för de data som ska returneras.
setData(mimeType, data)	Kopiera data till Urklipp. Ställ in parametern `mimeType` på MIME-typen för data.

JavaScript-kod som finns utanför sandlådan application kan bara få tillgång till Urklipp via dessa händelser. Innehåll i sandlådan application kan emellertid få tillgång till systemurklipp direkt via AIR-klassen Clipboard. Du kan till exempel använda följande programsats om du vill hämta textformatdata i Urklipp:

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

Giltiga MIME-typer för data är:

MIME-typ	Värde
Text	”text/plain”
HTML	”text/html”
URL	”text/uri-list”
Bitmapp	”image/x-vnd.adobe.air.bitmap”
Fillista	”application/x-vnd.adobe.air.file-list”

***Viktigt!** Enbart innehåll i sandlådan application kan få tillgång till fildata som finns i Urklipp. Ett säkerhetsfel uppstår om innehåll som inte är programinnehåll försöker komma åt ett filobjekt från Urklipp.*

Mer information om hur du använder Urklipp finns i ”Kopiera och klistra in” på sidan 597 och i Using the Pasteboard from JavaScript (Apple Developer Center).

Dra och släppa

Adobe AIR 1.0 och senare

Att dra och släppa till och från HTML skapar följande DOM-händelser: `dragstart`, `drag`, `dragend`, `dragenter`, `dragover`, `dragleave` och `drop`. Händelseobjektet som skickas i dessa händelser ger åtkomst till data som dras via egenskapen `dataTransfer`. Egenskapen `dataTransfer` refererar till ett objekt som omfattar samma metoder som objektet `clipboardData` som associeras med en urklippshändelse. Du kan till exempel använda följande funktion om du vill hämta textformatdata från en `drop`-händelse:

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

Objektet `dataTransfer` har följande viktiga medlemmar:

Medlem	Beskrivning
clearData(mimeType)	Rensar bort data. Ställ in parametern `mimeType` på MIME-typen för den datarepresentation som ska rensas.
getData(mimeType)	Hämta data som har dragits. Den här metoden kan bara anropas i en hanterare för händelsen `drop`. Ställ in parametern `mimeType` på MIME-typen för de data som ska hämtas.
setData(mimeType, data)	Ange de data som ska dras. Ställ in parametern `mimeType` på MIME-typen för data.
types	En array med strängar som innehåller MIME-typer för alla datarepresentationer som finns i objektet `dataTransfer`.
effectsAllowed	Anger om de data som dras kan kopieras, flyttas, länkas eller en kombination av dessa. Ställ in egenskapen `effectsAllowed` i hanteraren för händelsen `dragstart`.
dropEffect	Anger vilken av de tillåtna släpphändelserna som stöds av dragmålet. Ställ in egenskapen `dropEffect` i hanteraren för händelsen `dragEnter`. Under dragningen ändras markören till att ange vilken effekt som uppstår om användaren släpper musknappen. Om ingen `dropEffect`-effekt anges väljs en `effectsAllowed`-egenskapseffekt. Kopieringseffekten prioriteras över flytteffekten, som prioriteras över länkeffekten. Användaren kan ändra standardprioritet med hjälp av tangentbordet.

Mer information om hur du tillför stöd för dra och släpp i ett AIR-program finns i ”Dra och släppa i AIR” på sidan 609 och i Using the Drag-and-Drop from JavaScript (Apple Developer Center).

Egenskaperna innerHTML och outerHTML

Adobe AIR 1.0 och senare

I AIR gäller säkerhetsbegränsningar för hur egenskaperna `innerHTML` och `outerHTML` används för innehåll som körs i sandlådan application. Innan sidläsningshändelsen, och under körningen av alla inläsningshändelsehanterare, går det att använda egenskaperna `innerHTML` och `outerHTML` utan begränsningar. När sidan väl har lästs in kan du emellertid bara använda egenskapen `innerHTML` eller `outerHTML` för att lägga till statiskt innehåll i dokumentet. Alla programsatser i strängen som tilldelas `innerHTML` eller `outerHTML` som utvärderas till körbar kod ignoreras. Om du till exempel inkluderar ett händelseåterkopplingsattribut i en elementdefinition läggs händelseavlyssnaren inte till. Inbäddade `<script>`-taggar utvärderas inte heller. Du hittar mer information i ”HTML-säkerhet i Adobe AIR” på sidan 1085.

Metoderna Document.write() och Document.writeln()

Adobe AIR 1.0 och senare

Användningen av metoderna `write()` och `writeln()` är inte begränsad i sandlådan application före sidans `load`-händelse. När sidan väl har lästs in och du anropar någon av dessa metoder kommer ingen sida att rensas och ingen ny att skapas. I en annan sandlåda än application, liksom i de flesta webbläsare, rensas den aktuella sidan och en ny, tom öppnas om du anropar `document.write()` eller `writeln()` efter att sidan har lästs in.

Egenskapen Document.designMode

Adobe AIR 1.0 och senare

Ställ in egenskapen `document.designMode` på värdet `on` om du vill att alla element i dokumentet ska kunna redigeras. Det inbyggda redigeringsstödet omfattar textredigering, kopiera, klistra in samt dra och släpp. Att ställa in `designMode` på `on` motsvarar att ställa in egenskapen `contentEditable` för elementet `body` på `true`. Du kan använda egenskapen `contentEditable` för de flesta HTML-element om du vill definiera vilka avsnitt i ett dokument som ska kunna redigeras. Mer information finns i ”HTML-attributet contentEditable” på sidan 979.

unload-händelser (för body- och frameset-objekt)

Adobe AIR 1.0 och senare

I taggen `frameset` eller `body` på den översta nivån för ett fönster (bland annat programmets huvudfönster) ska du inte använda händelsen `unload` för att svara på fönstret (eller programmet) som stängs. I stället använder du händelsen `exiting` för objektet NativeApplication (för att identifiera när programmet stängs). Du kan också använda händelsen `closing` för objektet NativeWindow (för att identifiera när ett fönster stängs). I följande JavaScript-skriptkod visas meddelandet (`"Goodbye."`) när användaren stänger programmet:

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

Skript *kan* emellertid lyckas svara på `unload`-händelsen som orsakas av navigering för en bildruta, iframe eller fönsterinnehåll på den översta nivån.

Obs! *Dessa begränsningar kan komma att tas bort i framtida versioner av Adobe AIR.*

JavaScript-objektet Window

Adobe AIR 1.0 och senare

Window-objektet är fortfarande det globala objektet när det gäller JavaScript-körning. I sandlådan application tillför AIR nya egenskaper till JavaScript-objektet Window som ger tillgång till de inbyggda AIR-klasserna samt viktiga värdobjekt. Vissa metoder och egenskaper beter sig dessutom olika beroende på om de finns i sandlådan application eller inte.

Egenskapen Window.runtime Med egenskapen `runtime` kan du instansiera och använda inbyggda körningsklasser från sandlådan application. Dessa klasser omfattar AIR- och Flash Player-API:er (men inte till exempel Flex-ramverket). Följande programsatser skapar ett AIR-filobjekt:

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

Filen `AIRAliases.js`, som finns i AIR SDK, innehåller aliasdefinitioner som gör att du kan korta av sådana referenser. När till exempel `AIRAliases.js` importeras till en sida, kan ett File-objekt skapas med följande programsats:

```
var preferencesFile = new air.File();
```

Egenskapen `window.runtime` definieras bara för innehåll i sandlådan application och bara för det överordnade dokumentet på en sida med bildrutor och iframes.

Se ”Använda filen AIRAliases.js” på sidan 992.

Egenskapen Window.nativeWindow Egenskapen `nativeWindow` innehåller en referens till det underliggande inbyggda fönsterobjektet. Med den här egenskapen kan du skripta fönsterfunktioner och -egenskaper som skärmplacering, storlek och synlighet, och hantera fönsterhändelser som att stänga, ändra storlek på och flytta. I följande programsats stängs fönstret:

```
window.nativeWindow.close();
```

Obs! *De fönsterkontrollfunktioner som ingår i objektet NativeWindow överlappar funktionerna som finns i JavaScript-objektet Window. I sådana fall kan du välja den metod som passar bäst.*

Egenskapen `window.nativeWindow` definieras bara för innehåll i sandlådan application och bara för det överordnade dokumentet på en sida med bildrutor och iframes.

Egenskapen Window.htmlLoader Egenskapen `htmlLoader` innehåller en referens till AIR-objektet HTMLLoader som rymmer HTML-innehållet. Med den här egenskapen kan du skripta utseende och beteende i HTML-miljön. Du kan till exempel använda egenskapen `htmlLoader.paintsDefaultBackground` för att ta reda på om kontrollen målar en standardbakgrund i vitt:

```
window.htmlLoader.paintsDefaultBackground = false;
```

Obs! *Själva HTMLLoader-objektet har en `window`-egenskap som refererar till JavaScript-objektet Window för HTML-innehållet som det rymmer. Du kan använda den här egenskapen när du vill få åtkomst till JavaScript-miljön via en referens till det innehållande HTMLLoader.*

Egenskapen `window.htmlLoader` definieras bara för innehåll i sandlådan application och bara för det överordnade dokumentet på en sida med bildrutor och iframes.

Egenskaperna Window.parentSandboxBridge och Window.childSandboxBridge Med egenskaperna `parentSandboxBridge` och `childSandboxBridge` kan du definiera ett gränssnitt mellan en överordnad och underordnad bildruta. Mer information finns i ”Korsskriptning av innehåll i olika säkerhetssandlådor” på sidan 1003.

Funktionerna Window.setTimeout() och Window.setInterval() I AIR finns säkerhetsbegränsningar för hur funktionerna `setTimeout()` och `setInterval()` används i sandlådan application. Du kan inte definiera koden som ska köras som en sträng när `setTimeout()` eller `setInterval()` anropas. Du måste använda en funktionsreferens. Mer information finns i ”setTimeout() och setInterval()” på sidan 990.

Window.open(), funktion När metoden `open()` anropas av kod som körs i en annan sandlåda än application, öppnas bara ett fönster vid ett anrop som kommer från användarens åtgärder (till exempel klick med musen eller tryck på en tangent). Fönsterrubriken inleds dessutom med programnamnet (så att fönstret inte öppnas av fjärrinnehållet från imiterande fönster som öppnas av programmet). Mer information finns i ”Begränsningar för anrop till JavaScript-metoden window.open()” på sidan 1090.

Objektet air.NativeApplication

Adobe AIR 1.0 och senare

Objektet NativeApplication tillhandahåller information om programmets läge, skickar flera viktiga händelser på programnivå och tillhandahåller användbara funktioner för styrning av programmets beteende. En enskild instans av objektet NativeApplication skapas automatiskt och kan nås via den klassdefinierade egenskapen `NativeApplication.nativeApplication`.

Om du vill få tillgång till objektet från JavaScript-koden kan du använda:

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

Om `AIRAliases.js`-skriptet har importerats, kan du i stället använda det kortare formatet:

```
var app = air.NativeApplication.nativeApplication;
```

Objektet NativeApplication kan bara nås från sandlådan application. Mer information om objektet NativeApplication finns i ”Arbeta med körtids- och operativsystemsinformation i AIR” på sidan 890.

URL-schemat i JavaScript

Adobe AIR 1.0 och senare

Det går inte att köra kod som definieras i ett JavaScript URL-schema (som i `href="javascript:alert('Test')"`) i sandlådan application. Inget fel returneras.

HTML i AIR

Adobe AIR 1.0 och senare

AIR och WebKit definierar några HTML-element och -attribut som inte är standard, bland annat:

”Bildrute- och iframe-element i HTML” på sidan 976

”Händelsehanterare för HTML-element” på sidan 978

Bildrute- och iframe-element i HTML

Adobe AIR 1.0 och senare

AIR tillför nya attribut till bildrute- och iframe-elementen i innehållet i sandlådan application:

Attributet sandboxRoot Attributet `sandboxRoot` anger en alternativ, icke-programrelaterad domän för ursprunget till filen som anges med bildruteattributet `src`. Filen läses in i den icke-programrelaterade sandlåda som motsvarar angiven domän. Innehållet i filen och innehållet som läses in från angiven domän kan korsskripta varandra.

Viktigt! *Om du ställer in värdet för `sandboxRoot` till domänens bas-URL, läses alla förfrågningar om innehåll från den domänen in från programkatalogen i stället för fjärrservern (oavsett om förfrågningen resulterar från sidnavigering, från en XMLHttpRequest eller från någon annan metod för att läsa in innehåll).*

Attributet documentRoot Attributet `documentRoot` anger den lokala katalogen från vilken det ska läsas in URL:er som löses till filer på den plats som anges av `sandboxRoot`.

När URL:er löses, antingen i bildruteattributet `src` eller i innehåll som läses in till bildrutan, byts den del av URL:en som matchar värdet som anges i `sandboxRoot` ut mot värdet som anges i `documentRoot`. I följande bildrutetagg sker därför följande:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

`child.html` läses in från underkatalogen `sandbox` i programmets installationsmapp. Relativa URL:er i `child.html` löses utifrån katalogen `sandbox`. Observera att alla filer på fjärrservern på `www.example.com/air` inte är tillgängliga i bildrutan, eftersom AIR försöker att läsa in dem från katalogen app:/sandbox/.

Attributet allowCrossDomainXHR Ta med `allowCrossDomainXHR="allowCrossDomainXHR"` i den inledande bildrutetaggen om du vill tillåta att innehåll i bildrutan skickar XMLHttpRequests till en fjärrdomän. Innehåll som inte är programinnehåll kan som standard bara göra sådana förfrågningar till sin egen ursprungsdomän. Om du tillåter XHR:er mellan domäner kan det få allvarliga konsekvenser för säkerheten. Koden på sidan kan utbyta data med alla domäner. Om skadligt innehåll injiceras på sidan kan alla data som är tillgängliga för koden i den aktuella sandlådan påverkas. Du ska bara aktivera XHR:er mellan domäner för sidor som du skapar och kontrollerar och bara när datainläsning mellan domäner verkligen behövs. Du ska också noggrant kontrollera alla externa data som läses in av sidan för att förhindra kodinjiceringar eller andra former av attacker.

Viktigt! *Om attributet `allowCrossDomainXHR` ingår i ett bildrute- eller iframe-element aktiveras XHR:er mellan domäner (såvida inte värdet tilldelas ”0” eller börjar med bokstaven ”f” eller ”n”). Om du till exempel ställer in `allowCrossDomainXHR` på `”deny"` aktiveras ändå XHR:er mellan domäner. Ta inte med attributet över huvud taget i elementdeklarationen om du inte vill aktivera förfrågningar mellan domäner.*

Attributet ondominitialize Anger en händelsehanterare för händelsen `dominitialize` för en bildruta. Den här händelsen är en AIR-specifik händelse som utlöses när fönster- och dokumentobjekten för bildrutan har skapats, men innan några skript har analyserats eller dokumentelement har skapats.

Bildrutan skickar händelsen `dominitialize` tillräckligt tidigt i läsningssekvensen så att alla skript på den underordnade sidan kan referera till objekt, variabler och funktioner som har lagts till i det underordnade dokumentet av `dominitialize`-hanteraren. Den överordnade sidan måste finnas i samma sandlåda som den underordnade för att det ska gå att lägga till eller komma åt objekt direkt i ett underordnat dokument. En överordnad i sandlådan application kan emellertid skapa en sandlådebrygga för att kommunicera med innehåll i andra sandlådor än sandlådan application.

I följande exempel visas hur iframe-taggen används i AIR:

Placera `child.html` i en fjärrsandlåda, utan att mappa till en faktisk domän på en fjärrserver:

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

Placera `child.html` i en fjärrsandlåda, och tillåt bara XMLHttpRequests till `www.example.com`:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

Placera `child.html` i en fjärrsandlåda, och tillåt bara XMLHttpRequests till valfri fjärrdomän:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

Placera `child.html` i en lokal sandlåda i filsystemet:

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

Placera `child.html` i en fjärrsandlåda, där händelsen `dominitialize` används för att skapa en sandlådebrygga:

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

Följande `child.html`-dokument visar hur underordnat innehåll får åtkomst till den överordnade sandlådebryggan:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Mer information finns i ”Korsskriptning av innehåll i olika säkerhetssandlådor” på sidan 1003 och ”HTML-säkerhet i Adobe AIR” på sidan 1085.

Händelsehanterare för HTML-element

Adobe AIR 1.0 och senare

DOM-objekt i AIR och WebKit skickar vissa händelser, som inte finns i den vanliga DOM-händelsemodellen. I tabellen nedan anges vilka närliggande händelseattribut som du kan använda för att ange hanterare för dessa händelser:

Namn på återanropsattribut	Beskrivning
oncontextmenu	Anropas när en snabbmeny anropas, till exempel via ett högerklick eller kommando-klick på markerad text.
oncopy	Anropas när en markering i ett element kopieras.
oncut	Anropas när en markering i ett element klipps ut.
ondominitialize	Anropas när DOM för ett dokument som har lästs in i en bildruta eller iframe skapas, men innan några DOM-element skapas eller skript analyseras.
ondrag	Anropas när ett element dras.
ondragend	Anropas när någon slutar att dra.

Namn på återanropsattribut	Beskrivning
ondragenter	Anropas när någon drar innanför gränsen till ett element.
ondragleave	Anropas när någon drar utanför gränsen till ett element.
ondragover	Anropas fortlöpande när någon drar inom gränserna till ett element.
ondragstart	Anropas när en dragning inleds.
ondrop	Anropas när en dragning släpps över ett element.
onerror	Anropas när ett fel inträffar medan ett element läses in.
oninput	Anropas när text skrivs i ett formulärelementet.
onpaste	Anropas när ett objekt klistras in i ett element.
onscroll	Anropas när innehållet i ett rullningsbart element rullas.
onselectstart	Anropas när en markering börjar.

HTML-attributet contentEditable

Adobe AIR 1.0 och senare

Du kan lägga till attributet `contentEditable` till valfritt HTML-element så att användarna kan redigera innehållet i elementet. I följande exempel på HTML-kod ställs hela dokumentet in på att vara redigerbart, förutom det första `p`-elementet:

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

Obs! *Om du ställer in egenskapen `document.designMode` på `on` kan alla element i dokumentet redigeras, oavsett inställningen av `contentEditable` för ett enskilt element. Om du ställer in `designMode` på `off` inaktiveras emellertid inte redigering av element där `contentEditable` är `true`. Mer information finns i ”Egenskapen Document.designMode” på sidan 974.*

Data: URL

Adobe AIR 2 och senare

AIR stöder konstruktionen `data:` URL för följande element:

- img
- indatatyp=”bild”
- CSS-regler som tillåter bilder (till exempel bakgrundsbilder)

Med hjälp av data-URL:er kan du infoga binära bilddata direkt i ett CSS- eller HTML-dokument som en base64-kodad sträng. I följande exempel används data: URL som en repeterande bakgrund:

```
<html>
<head>
<style>
body {
background-
image:url('
0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAA
BF0lEQVR42uzZQQ7CMAxE0e%2F7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShX
BgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0FRmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCd
m1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBctAKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bS
qHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMiQsSFiQ8vRscTjlQOI2iHZwtpHuf
%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

Tänk på att det är viktigt att lämna plats för extra utrymme när du använder data: URL. Datasträngen måste till exempel anges på en enda obruten rad. Annars behandlas radbrytningarna som del av data och bilden kan inte avkodas.

CSS i AIR

Adobe AIR 1.0 och senare

WebKit stöder flera utökade CSS-egenskaper. Många av dessa tillägg använder prefixet: `-webkit`. Observera att vissa av dessa tillägg är experimentella till sin natur och kan komma att tas bort från framtida versioner av WebKit. Mer information om Webkit-stöd för CSS och de olika tilläggen till CSS finns i Safari CSS Reference.

WebKit-funktioner som inte stöds i AIR

Adobe AIR 1.0 och senare

AIR stöder inte följande funktioner tillgängliga i WebKit eller Safari 4:

- Meddelanden mellan domäner via window.postMessage (AIR har egna API:er för kommunikation mellan domäner)
- CSS-variabler
- WOFF (Web Open Font Format) och SVG-teckensnitt.
- Taggar för HTML-video och -ljud
- Mediaenhetsfrågor
- Offline-programcache
- Utskrift (AIR har ett eget PrintJob-API)
- Stavnings- och grammatikkontroll
- SVG
- WAI-ARIA

- WebSockets (AIR har egna socket-API:er)
- Web Workers
- WebKit SQL API (AIR innehåller ett eget API)
- WebKit-API för geopositionering (AIR innehåller ett eget API för geopositionering på de enheter som stöds)
- WebKit-API för överföring av flera filer
- WebKit-pekhändelser (AIR innehåller egna pekhändelser)
- Wireless Markup Language (WML)

Följande lista innehåller specifika JavaScript API, HTML-element och CSS-egenskaper och -värden som inte stöds i AIR:

Fönsterobjektmedlemmar i JavaScript som inte stöds:

- applicationCache()
- console
- openDatabase()
- postMessage()
- document.print()

HTML-taggar som inte stöds:

- audio
- video

HTML-attribut som inte stöds:

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate
- ontouchstart
- ontouchmove
- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern

- required
- sandbox

JavaScript-händelser som inte stöds:

- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove
- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

CSS-egenskaper som inte stöds:

- background-clip
- background-origin (använd -webkit-background-origin)
- background-repeat-x
- background-repeat-y
- background-size (använd -webkit-background-size)
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip
- -webkit-color-correction
- -webkit-font-smoothing

CSS-värden som inte stöds:

- egenskapsvärden för appearance:
  - media-volume-slider-container
  - media-volume-slider
  - media-volume-sliderthumb

  - outer-spin-button
- border-box (background-clip och background-origin)
- contain (background-size)
- content-box (background-clip och background-origin)
- cover (background-size)
- egenskapsvärden för list:
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem
  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant
  - lower-norwegian
  - oromo
  - sidama
  - somali
  - tigre
  - tigrinya-er
  - tigrinya-er-abegede
  - tigrinya-et

  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee (visningsegenskap)

Kapitel 59: Programmera HTML och JavaScript i AIR

Adobe AIR 1.0 och senare

Det finns ett antal programmeringsområden som är unika när det gäller att utveckla Adobe® AIR®-program med hjälp av HTML och JavaScript. Följande information är viktig om du programmerar ett HTML-baserat AIR-program eller ett SWF-baserat AIR-program som kör HTML och JavaScript med hjälp av klassen HTMLLoader (eller komponenten mx:HTML Flex™).

Om klassen HTMLLoader

Adobe AIR 1.0 och senare

Klassen HTMLLoader i Adobe AIR används för att definiera visningsobjektet som kan visa HTML-innehåll i ett AIR-program. SWF-baserade program kan lägga till en HTMLLoader-kontroll i ett befintligt fönster eller skapa ett HTML-fönster som automatiskt innehåller ett HTMLLoader-objekt med `HTMLLoader.createRootWindow()`. HTMLLoader-objektet kan nås via JavaScript-egenskapen `window.htmlLoader` i den inlästa HTML-sidan.

Läsa in HTML-innehåll från en URL

Adobe AIR 1.0 och senare

Följande kod läser in en URL i ett HTMLLoader-objekt (lägg till HTMLLoader-objektet som ett underordnat objekt till scenen eller till en annan behållare för visningsobjekt så att HTML-innehållet visas i programmet):

```
import flash.html.HTMLLoader;

var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
```

Egenskaperna `width` och `height` i ett HTMLLoader-objekt är som standard inställda på 0. Du måste ange de här måtten när du lägger till ett HTMLLoader-objekt i scenen. HTMLLoader skickar flera händelser när en sida läses in. Du kan använda de här händelserna för att avgöra när det går bra att interagera med den inlästa sidan. De här händelserna beskrivs i ”Hantera HTML-relaterade händelser i AIR” på sidan 1025.

Obs! *I Flex-ramverket kan bara klasser som utökar klassen UIComponent läggas till som underordnade komponenter i en Flex-behållare. På grund av detta kan du inte lägga till ett HTMLLoader-objekt direkt som en underordnad komponent i en Flex-behållare. Du kan däremot använda Flex-kontrollen mx:HTML, skapa en anpassad klass som utökar UIComponent och innehåller ett HTMLLoader-objekt som är underordnat UIComponent och lägga till HTMLLoader som underordnat en UIComponent och lägga till UIComponent i Flex-behållaren.*

Du kan också återge HTML-text med hjälp av klassen TextField, men dess funktioner är begränsade. Klassen TextField i Adobe® Flash® Player har stöd för en delmängd HTML-kod, men på grund av storleksbegränsningar har inte klassen så många funktioner. (Klassen HTMLLoader som ingår i Adobe AIR är inte tillgänglig i Flash Player.)

Läsa in HTML-innehåll från en sträng

Adobe AIR 1.0 och senare

Metoden `loadString()` i ett HTMLLoader-objekt läser in en sträng med HTML-innehåll i HTMLLoader-objektet:

```
var html:HTMLLoader = new HTMLLoader();
var htmlStr:String = "<html><body>Hello <b>world</b>.</body></html>";
html.loadString(htmlStr);
```

Som standard placeras innehåll som du läser in med metoden `loadString()` i en icke-programsandlåda med följande egenskaper:

- Innehåll kan läsas in från nätverket (men inte från filsystemet).
- Data kan inte läsas in med hjälp av XMLHttpRequest.
- Egenskapen `window.location` har värdet `"about:blank"`.
- Innehållet har inte åtkomst till egenskapen `window.runtime` (som innehåll i andra icke-programsandlådor har).

I AIR 1.5 innehåller HTMLLoader-klassen egenskapen `placeLoadStringContentInApplicationSandbox`. När den här egenskapen har värdet `true` för ett HTMLLoader-objekt placeras innehåll som du läser in med metoden `loadString()` i programsandlådan. (Standardvärdet är `false`.) Det innebär att innehåll som lästs in med metoden `loadString()` har åtkomst till egenskapen `window.runtime` och till alla AIR-API:er. Om du ändrar värdet för den här egenskapen till `true` kontrollerar du att datakällan för strängen som används i anrop till `loadString()`-metoden är tillförlitlig. Kod i HTML-strängen körs med fullständig programbehörighet när den här egenskapen har värdet `true`. Använd bara värdet `true` för den här egenskapen om du är helt säker på att strängen inte kan innehålla skadlig kod.

I program som skapats med SDK-paket för AIR 1.0 eller AIR 1.1 placeras innehåll som du läser in med `loadString()`-metoden i programsandlådan.

Viktiga säkerhetsregler vid användning av HTML i AIR-program

Adobe AIR 1.0 och senare

De filer som du installerar med AIR-programmet har åtkomst till programmeringsgränssnitten i AIR. Av säkerhetsskäl gäller inte detta innehåll från andra källor. Den här begränsningen förhindrar till exempel innehåll från en fjärrdomän (t.ex. http://exempel.com) från att läsa innehållet i användarens skrivbordskatalog (eller ännu värre).

Eftersom det finns säkerhetsluckor som kan utnyttjas genom att anropa funktionen `eval()` (och tillhörande programmeringsgränssnitt), så är det innehåll som installeras tillsammans med programmet som standard begränsat och kan inte använda de här metoderna. Vissa Ajax-ramverk använder dock anrop av funktionen `eval()` och tillhörande programmeringsgränssnitt.

För att du ska kunna strukturera innehåll som ska användas och fungera i ett AIR-program, måste du ta hänsyn till de regler för säkerhetsbegränsning som finns för innehåll från olika källor. Innehåll från olika källor placeras i separata säkerhetsklasser, så kallade sandlådor (se ”Säkerhetssandlådor” på sidan 1049). Innehåll som installeras med programmet installeras som standard i en sandlåda som kallas *program*sandlådan och det gör att innehållet får åtkomst till programmeringsgränssnitten i AIR. Programsandlådan är vanligtvis den säkraste sandlådan och dess begränsningar har utformats för att förhindra att otillförlitlig kod körs.

Runtime-modulen tillåter att du läser in innehåll som installerats med programmet i en annan sandlåda än programsandlådan. Innehåll i icke-programsandlådor används i en säkerhetsmiljö som liknar den som finns i en vanlig webbläsare. Kod i icke-programsandlådor kan till exempel använda `eval()` och andra relaterade metoder (men har ändå inte åtkomst till programmeringsgränssnitten i AIR). I runtime-modulen finns olika sätt att få innehåll i olika sandlådor att kommunicera på ett säkert sätt (utan att till exempel exponera programmeringsgränssnitt i AIR för icke-programinnehåll). Mer information finns i ”Korsskriptning av innehåll i olika säkerhetssandlådor” på sidan 1003.

Om du anropar kod som är begränsad från användning i en sandlåda av säkerhetsskäl skickar runtime-modulen ett JavaScript-fel av liknande typ: ”Säkerhetsöverträdelse i Adobe AIR vid körning av JavaScript-kod i programmets säkerhetssandlåda.”

Om du vill undvika det här felet följer du de kodningsrutiner som beskrivs i nästa avsnitt, ”Undvika säkerhetsrelaterade JavaScript-fel” på sidan 987.

Du hittar mer information i ”HTML-säkerhet i Adobe AIR” på sidan 1085.

Undvika säkerhetsrelaterade JavaScript-fel

Adobe AIR 1.0 och senare

Om du anropar kod som är begränsad från användning i en sandlåda på grund av de här säkerhetsreglerna skickar runtime-modulen ett JavaScript-fel av liknande typ: ”Säkerhetsöverträdelse i Adobe AIR vid körning av JavaScript-kod i programmets säkerhetssandlåda.” Följ de här kodningsrutinerna om du vill undvika det här felet.

Orsaker till säkerhetsrelaterade JavaScript-fel

Adobe AIR 1.0 och senare

Kod som körs i programsandlådan är begränsad för de flesta åtgärder som innebär utvärdering och körning av strängar när dokumentets `load`-händelse har utlösts och eventuella `load`-händelsehanterare har avslutats. Försök att använda följande typer av JavaScript-programsatser som utvärderar och kör potentiellt osäkra strängar genererar JavaScript-fel:

- Funktionen eval()
- setTimeout() och setInterval()
- Funktionskonstruktor

  Dessutom misslyckas följande typer av JavaScript-programsatser utan att generera fel om osäker JavaScript:

- javascript: URL:er
- Händelseåterkopplingar som är tilldelade via onevent-attribut i innerHTML- och outerHTML-programsatser
- Inläsning av JavaScript-filer från plats utanför programmets installationskatalog
- document.write() och document.writeln()
- Synkrona XMLHttpRequests innan load-händelsen eller under en load-händelsehanterare
- Dynamiskt skapade skriptelement

  Obs! *I vissa begränsningsfall tillåts utvärdering av strängar. Se ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087 om du vill ha mer information.*

Adobe har en lista över Ajax-ramverk som kan hantera programmets säkerhetssandlåda. Den finns på http://www.adobe.com/go/airappsandboxframeworks_se.

Följande avsnitt beskriver hur du skriver om skript så att du kan undvika de här osäkra JavaScript-felen och tysta fel för kod som körs i programsandlådan.

Mappa programinnehåll till en annan sandlåda

Adobe AIR 1.0 och senare

I de flesta fall kan du skriva över eller strukturera om ett program så att säkerhetsrelaterade JavaScript-fel undviks. Om omskrivning eller omstrukturering inte är möjlig kan du dock läsa in programinnehållet i en annan sandlåda med hjälp av den teknik som beskrivs i ”Läsa in programinnehåll i en icke-programsandlåda” på sidan 1004. Om det här innehållet också måste ha tillgång till AIR-programmeringsgränssnitten kan du skapa en sandlådebrygga enligt beskrivningen i ”Konfigurera ett gränssnitt för en sandlådebrygga” på sidan 1005.

Funktionen eval()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I programsandlådan kan bara funktionen `eval()` användas innan sidans `load`-händelse eller under en `load`-händelsehanterare. När sidan har lästs in kommer inte anrop till `eval()` att köra koden. I följande fall kan du dock skriva om koden så att den undviker användning av `eval()`.

Tilldela egenskaper till ett objekt

Adobe AIR 1.0 och senare

I stället för att tolka en sträng för att skapa egenskapskopplaren:

```
eval("obj." + propName + " = " + val);
```

använd egenskaper med hakparenteser:

```
obj[propName] = val;
```

Skapa en funktion med variabler som finns i kontexten

Adobe AIR 1.0 och senare

Ersätt programsatser enligt följande:

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

med:

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

Skapa ett objekt genom att använda namnet på klassen som en strängparameter

Adobe AIR 1.0 och senare

Ta till exempel en hypotetisk JavaScript-klass som definierats med följande kod:

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

Det enklaste sättet att skapa en instans är att använda `eval()`:

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

Du kan dock undvika anrop till `eval()` genom att tolka varje komponent av klassnamnet och skapa det nya objektet med hjälp av hakparenteser:

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

Om du vill skapa instansen använder du:

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

setTimeout() och setInterval()

Adobe AIR 1.0 och senare

Ersätt den sträng som skickas som hanterarfunktion med en funktionsreferens eller ett objekt. Ersätt till exempel en programsats som:

```
setTimeout("alert('Timeout')", 100);
```

med:

```
setTimeout(function(){alert('Timeout')}, 100);
```

Om funktionen kräver att objektet `this` är angett av anroparen kan du ersätta en programsats som:

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

med följande:

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

Funktionskonstruktor

Adobe AIR 1.0 och senare

Anrop till `new Function(param, body)` kan ersättas med en textbunden funktionsdeklaration eller bara användas innan sidans `load`-händelse har hanterats.

javascript: URL:er

Adobe AIR 1.0 och senare

Koden som definierats i en länk med hjälp av javascript: URL-schemat ignoreras i programsandlådan. Inget fel om osäker JavaScript genereras. Du kan ersätta länkar med hjälp av javascript: URL:er som:

```
<a href="javascript:code()">Click Me</a>
```

med:

```
<a href="#" onclick="code()">Click Me</a>
```

Händelseåterkopplingar som är tilldelade via onevent-attribut i innerHTML- och outerHTML-programsatser

Adobe AIR 1.0 och senare

När du använder innerHTML eller outerHTML för att lägga till element i ett dokuments DOM, ignoreras alla händelseåterkopplingar som tilldelats i programsatsen, t.ex. `onclick` och `onmouseover`. Inget säkerhetsfel genereras. Du kan i stället tilldela ett `id`-attribut till de nya elementen och ställa in så att händelsehanterarens återkopplingsfunktioner använder metoden `addEventListener()`.

Ta till exempel ett angivet målelement i ett dokument, såsom:

```
<div id="container"></div>
```

Ersätt programsatser som:

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

med:

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

Inläsning av JavaScript-filer från plats utanför programmets installationskatalog

Adobe AIR 1.0 och senare

Det är inte tillåtet att läsa in skriptfiler från en plats utanför programsandlådan. Inget säkerhetsfel genereras. Alla skriptfiler som körs i programsandlådan måste vara installerade i programkatalogen. Om du vill använda externa skript på en sida måste du mappa sidan till en annan sandlåda. Se ”Läsa in programinnehåll i en icke-programsandlåda” på sidan 1004.

document.write() och document.writeln()

Adobe AIR 1.0 och senare

Anrop till `document.write()` och `document.writeln()` ignoreras efter att sidans `load`-händelse har hanterats. Inget säkerhetsfel genereras. Du kan också läsa in en ny fil eller ersätta innehållet i dokumentet med hjälp av DOM-ändringstekniker.

Synkrona XMLHttpRequests innan load-händelsen eller under en load-händelsehanterare

Adobe AIR 1.0 och senare

Synkrona XMLHttpRequests som startats innan sidans `load`-händelse eller under en `load`-händelsehanterare returnerar inget innehåll. Asynkrona XMLHttpRequests kan startas men returneras inte förrän efter `load`-händelsen. När `load`-händelsen har hanterats beter sig synkrona XMLHttpRequests normalt.

Dynamiskt skapade skriptelement

Adobe AIR 1.0 och senare

Dynamiskt skapade skriptelement som t.ex. är skapade med innerHTML eller metoden `document.createElement()` ignoreras.

Åtkomst till programmeringsgränssnittsklasser i AIR från JavaScript

Adobe AIR 1.0 och senare

Förutom standardelementen och de utökade elementen i Webkit, kan HTML- och JavaScript-kod få åtkomst till värdklasserna från runtime-modulen. Med hjälp av de här klasserna får du tillgång till avancerade funktioner i AIR, t.ex.

- Tillgång till filsystemet
- Använda lokala SQL-databaser
- Kontrollera menyer i fönster och program
- Tillgång till sockets för nätverk
- Använda användardefinierade klasser och objekt
- Ljudfunktioner

  I AIR-filens programmeringsgränssnitt finns till exempel en File-klass som finns i flash.filesystem-paketet. Du kan skapa ett File-objekt i JavaScript på följande sätt:

  ```
  var myFile = new window.runtime.flash.filesystem.File();
  ```

  Objektet `runtime` är ett särskilt JavaScript-objekt som är tillgängligt för HTML-innehåll som körs i programsandlådan i AIR. Det ger dig åtkomst till runtime-klasser från JavaScript. `flash`-egenskapen i `runtime`-objektet ger åtkomst till flash-paketet. `flash.filesystem`-egenskapen i `runtime`-objektet ger i sin tur åtkomst till flash.filesystem-paketet (och i det här paketet ingår klassen File). Paket är ett sätt att ordna klasser som används i ActionScript.

  Obs! *`runtime`-egenskapen läggs inte automatiskt till i fönsterobjekten för de sidor som blir inlästa i ett frame- eller iframe-element. Så länge som det underordnade dokumentet är i programsandlådan får det åtkomst till `runtime`-egenskapen i det överordnade dokumentet.*

  Eftersom paketstrukturen för runtime-klasserna skulle kräva att utvecklarna skrev väldigt långa strängar med JavaScript-kod för att få åtkomst till varje klass (som i `window.runtime.flash.desktop.NativeApplication`), så omfattar AIR SDK en AIRAliases.js-fil som gör att du får åtkomst till runtime-klasserna på ett mycket enklare sätt (t.ex. genom att skriva `air.NativeApplication`).

  Programmeringsgränssnittsklasserna i AIR beskrivs i den här handboken. Övriga klasser från API:t för Flash Player, som kan vara intressanta för HTML-utvecklare, beskrivs i *API-referens för Adobe AIR för HTML-utvecklare*. ActionScript är det språk som används i SWF-innehåll (Flash Player). Syntaxen för JavaScript och ActionScript är dock väldigt lika. (Båda baseras på versioner av ECMAScript-språket.) Alla inbyggda klasser finns i både JavaScript (i HTML-innehåll) och ActionScript (i SWF-innehåll).

  Obs! *JavaScript-kod kan inte använda klasserna Dictionary, XML och XMLList som finns i ActionScript.*

Använda filen AIRAliases.js

Adobe AIR 1.0 och senare

Runtime-klasserna är ordnade i en paketstruktur på följande sätt:

- `window.runtime.flash.desktop.NativeApplication`

- `window.runtime.flash.desktop.ClipboardManager`
- `window.runtime.flash.filesystem.FileStream`
- `window.runtime.flash.data.SQLDatabase`

AIR SDK innehåller en AIRAliases.js-fil som tillhandahåller ”alias”-definitioner som ger dig åtkomst till runtime-klasserna utan att du behöver skriva så mycket. Du kan till exempel få åtkomst till klasserna ovan genom att skriva följande:

- `air.NativeApplication`
- `air.Clipboard`
- `air.FileStream`
- `air.SQLDatabase`

Den här listan är bara en liten del av alla de klasser som finns i AIRAliases.js-filen. En fullständig lista över klasser och funktioner på paketnivå finns i *API-referens för Adobe AIR för HTML-utvecklare*.

Förutom ofta använda runtime-klasser innehåller AIRAliases.js-filen även alias för vanliga funktioner på paketnivå: `window.runtime.trace()`, `window.runtime.flash.net.navigateToURL()` och `window.runtime.flash.net.sendToURL()`, som har alias `air.trace()`, `air.navigateToURL()` och `air.sendToURL()`.

Om du vill använda AIRAliases.js-filen inkluderar du följande `script`-referens på HTML-sidan:

```
<script src="AIRAliases.js"></script>
```

Justera sökvägen i `src`-referens efter behov.

Viktigt! *JavaScript-exempelkoden i den här dokumentationen förutsätter att du har inkluderat AIRAliases.js-filen på HTML-sidan, om inget annat anges.*

Om URL:er i AIR

Adobe AIR 1.0 och senare

I HTML-innehåll som körs i AIR kan du använda något av följande URL-scheman när du definierar `src`-attribut för `img`-, `frame`-, `iframe`- och `script`-taggar, i `href`-attribut i en `link`-tagg eller på andra ställen där du kan ange en URL.

URL-schema	Beskrivning	Exempel
file	En sökväg som är relativ till filsystemets rot.	`file:///c:/AIR Test/test.txt`
app	En sökväg som är relativ till det installerade programmets rotkatalog.	`app:/images`
app-storage	En sökväg som är relativ till programlagringskatalogen. För varje installerat program definieras en unik programlagringskatalog i AIR, där du kan lagra data för just det programmet.	`app-storage:/settings/prefs.xml`
http	En vanlig HTTP-begäran.	`http://www.adobe.com`
https	En vanlig HTTPS-begäran.	`https://secure.example.com`

Mer information om hur du använder URL-scheman i AIR finns i ”URI-scheman” på sidan 815.

Många AIR-programmeringsgränssnitt, inklusive klasserna File, Loader, URLStream och Sound, använder ett URLRequest-objekt i stället för en sträng som innehåller URL:en. Själva URLRequest-objektet initieras med en sträng som kan använda något av dessa url-scheman. Följande programsats skapar till exempel ett URLRequest-objekt som kan användas för att begära Adobes hemsida:

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

Mer information om URLRequest-objekt finns i ”HTTP-kommunikation” på sidan 813.

Göra ActionScript-objekt tillgängliga för JavaScript

Adobe AIR 1.0 och senare

JavaScript på en HTML-sida som blir inläst av ett HTMLLoader-objekt kan anropa de klasser, objekt och funktioner som är definierade i ActionScript-körningskontexten med hjälp av egenskaperna `window.runtime`, `window.htmlLoader` och `window.nativeWindow` för HTML-sidan. Du kan också göra ActionScript-objekt och -funktioner tillgängliga för JavaScript-kod genom att skapa referenser till dem i JavaScript-körningskontexten.

Ett enkelt exempel på hur du kan få åtkomst till JavaScript-objekt från ActionScript

Adobe AIR 1.0 och senare

Följande exempel visar hur du kan lägga till egenskaper som refererar till ActionScript-objekt i det globala window-objektet för en HTML-sida:

```
var html:HTMLLoader = new HTMLLoader();
var foo:String = "Hello from container SWF."
function helloFromJS(message:String):void {
    trace("JavaScript says:", message);
}
var urlReq:URLRequest = new URLRequest("test.html");
html.addEventListener(Event.COMPLETE, loaded);
html.load(urlReq);

function loaded(e:Event):void{
    html.window.foo = foo;
    html.window.helloFromJS = helloFromJS;
}
```

HTML-innehållet (i en fil som heter test.html) som blir inläst i HTMLLoader-objektet i föregående exempel får tillgång till den `foo`-egenskap och `helloFromJS()`-metod som är definierad i den överordnade SWF-filen:

```
<html>
    <script>
        function alertFoo() {
            alert(foo);
        }
    </script>
    <body>
        <button onClick="alertFoo()">
            What is foo?
        </button>
        <p><button onClick="helloFromJS('Hi.')">
            Call helloFromJS() function.
        </button></p>
    </body>
</html>
```

När du får åtkomst till JavaScript-kontexten i ett dokument som blir inläst, kan du använda `htmlDOMInitialize`-händelsen för att skapa objekt väldigt tidigt i sidkonstruktionen så att de skript som definieras på sidan kan få åtkomst till dem. Om du väntar på `complete`-händelsen kan bara skripten som körs efter sidans `load`-händelse få åtkomst till de tillagda objekten.

Göra klassdefinitioner tillgängliga för JavaScript

Adobe AIR 1.0 och senare

Om du vill göra ActionScript-klasserna i programmet tillgängliga i JavaScript, kan du tilldela det inlästa HTML-innehållet till den programdomän som innehåller klassdefinitionerna. Programdomänen för JavaScript-körningskontexten kan anges med hjälp av `runtimeApplicationDomain`-egenskapen för HTMLLoader-objektet. Om du vill ange programdomänen som den primära programdomänen, anger du till exempel `runtimeApplicationDomain` som `ApplicationDomain.currentDomain` enligt följande kod:

```
html.runtimeApplicationDomain = ApplicationDomain.currentDomain;
```

När `runtimeApplicationDomain`-egenskapen är angiven delar JavaScript-kontexten klassdefinitioner med den tilldelade domänen. Om du vill skapa en instans av en anpassad klass i JavaScript, refererar du klassdefinitionen via `window.runtime`-egenskapen och använder operatorn `new`:

```
var customClassObject = new window.runtime.CustomClass();
```

HTML-innehållet måste vara från en kompatibel säkerhetsdomän. Om HTML-innehållet är från en annan säkerhetsdomän än den som tillhör den tilldelade programdomänen, använder sidan en standardprogramdomän i stället. Om du till exempel läser in en fjärrsida från Internet kan du inte tilldela ApplicationDomain.currentDomain som sidans programdomän.

Ta bort händelseavlyssnare

Adobe AIR 1.0 och senare

Om du lägger till JavaScript-händelseavlyssnare till objekt utanför den aktuella sidan, inklusive runtime-objekt, objekt i inläst SWF-innehåll och JavaScript-objekt som körs på andra sidor, måste du alltid ta bort de här händelseavlyssnarna när sidan tas bort. Annars skickar händelseavlyssnaren händelsen till en hanterarfunktion som inte finns längre. Om det här inträffar visas ett felmeddelande av liknande typ: ”Programmet försökte referera till ett JavaScript-objekt på en HTML-sida som inte finns inläst.” Om du tar bort onödiga händelseavlyssnare frigörs det tillhörande minnet och AIR återfår det. Mer information finns i ”Ta bort händelseavlyssnare på HTML-sidor som innehåller navigering” på sidan 1030.

Åtkomst till HTML DOM och JavaScript-objekt från ActionScript

Adobe AIR 1.0 och senare

När HTMLLoader-objektet skickar `complete`-händelsen kan du få åtkomst till alla objekt i HTML DOM (Document Object Model) för sidan. De tillgängliga objekten omfattar visningselement (t.ex. `div`- och `p`-objekt på sidan) samt JavaScript-variabler och -funktioner. `complete`-händelsen motsvarar JavaScript-sidans `load`-händelse. DOM-element, variabler och funktioner har kanske inte tolkats eller skapats innan `complete` har skickats. Om det är möjligt kan du vänta på `complete`-händelsen innan du får tillgång till HTML DOM.

Ta till exempel följande HTML-sida:

```
<html>
    <script>
        foo = 333;
        function test() {
            return "OK.";
        }
    </script>
    <body>
        <p id="p1">Hi.</p>
    </body>
</html>
```

På den här enkla HTML-sidan definieras en JavaScript-variabel som heter *foo* och en JavaScript-funktion som heter *test()*. De här båda är egenskaper för sidans globala `window`-objekt. `window.document`-objektet innehåller också ett namngivet P-element (med ID *p1*) som du kan få tillgång till med hjälp av metoden `getElementById()`. När sidan har lästs in (när HTMLLoader-objektet skickar `complete`-händelsen) får du tillgång till vart och ett av de här objekten från ActionScript. Se följande ActionScript-kod:

```
var html:HTMLLoader = new HTMLLoader();
html.width = 300;
html.height = 300;
html.addEventListener(Event.COMPLETE, completeHandler);
var xhtml:XML =
    <html>
        <script>
            foo = 333;
            function test() {
                return "OK.";
            }
        </script>
        <body>
            <p id="p1">Hi.</p>
        </body>
    </html>;
html.loadString(xhtml.toString());

function completeHandler(e:Event):void {
    trace(html.window.foo); // 333
    trace(html.window.document.getElementById("p1").innerHTML); // Hi.
    trace(html.window.test()); // OK.
}
```

Om du vill få tillgång till innehållet i ett HTML-element använder du egenskapen `innerHTML`. Den föregående koden använder till exempel `html.window.document.getElementById("p1").innerHTML` för att hämta innehållet i HTML-elementet med namnet *p1*.

Du kan också ange egenskaper för HTML-sidan från ActionScript. Följande exempel anger till exempel innehållet för `p1`-elementet och värdet på JavaScript-variabeln `foo` på sidan med hjälp av en referens till det HTMLLoader-objekt som ingår:

```
html.window.document.getElementById("p1").innerHTML = "Goodbye";
html.window.foo = 66;
```

Bädda in SWF-innehåll i HTML

Adobe AIR 1.0 och senare

Du kan bädda in SWF-innehåll i HTML-innehåll i ett AIR-program på samma sätt som i en webbläsare. Bädda in SWF-innehållet med hjälp av en `object`-tagg, en `embed`-tagg eller båda två.

Obs! *En vanlig webbkodningsmetod är att använda både en `object`-tagg och en `embed`-tagg för att visa SWF-innehåll på en HTML-sida. Det är ingen fördel att använda den här metoden i AIR. Du kan använda W3C-standardens `object`-tagg i innehåll som ska visas i AIR. Du kan förstås fortsätta att använda `object`- och `embed`-taggarna tillsammans, om så behövs, för HTML-innehåll som också visas i en webbläsare.*

Om du har aktiverat genomskinlighet i objektet NativeWindow där HTML- och SWF-innehållet visas, så visas inte SWF-innehållet i AIR när fönsterläget (`wmode`) som används för att bädda in innehållet är inställt på värdet `window`. Om du vill visa SWF-innehåll på en HTML-sida i ett genomskinligt fönster ställer du in parametern `wmode` på `opaque` eller `transparent`. Värdet `window` är standardinställning för `wmode`, så om du inte anger ett värde kanske inte innehållet visas.

Följande exempel visar hur `object`-taggen i HTML används för att visa en SWF-fil i HTML-innehåll. Parametern `wmode` är inställd på `opaque` så att innehållet visas, även om det underliggande NativeWindow-objektet är genomskinligt. SWF-filen blir inläst från programkatalogen men du kan använda ett URL-schema som stöds av AIR. (Den plats som SWF-filen blir inläst från bestämmer i vilken säkerhetssandlåda som AIR placerar innehållet.)

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

Du kan även använda ett skript för att läsa in innehåll dynamiskt. I följande exempel skapas en `object`-nod för att visa den SWF-fil som angetts i parametern `urlString`. I exemplet läggs noden till som underordnad i sidelementet med det ID som angetts av parametern `elementID`:

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

SWF-innehåll visas inte om objektet HTMLLoader är skalat eller roterat, eller om egenskapen `alpha` är inställd på ett annat värde än 1.0. Före AIR 1.5.2 visades inte SWF-innehåll i ett genomskinligt fönster oavsett vilket `wmode`-värde som var angivet.

Obs! *När ett inbäddat SWF-objekt försöker läsa in en extern resurs, som en videofil, kan det hända att SWF-innehållet inte återges korrekt om ingen absolut sökväg till videofilen anges i HTML-filen. Ett inbäddat SWF-objekt kan däremot läsa in en extern bildfil med en relativ sökväg.*

Följande exempel visar hur externa resurser kan läsas in via ett SWF-objekt som bäddats in i HTML-innehåll:

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

I följande ActionScript-exempel läses den bildsökväg som skickas av HTML-filen och bilden läses in på scenen:

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

Använda ActionScript-bibliotek i en HTML-sida

Adobe AIR 1.0 och senare

I AIR utökas HTML-skriptelementet så att en sida kan importera ActionScript-klasser i en kompilerad SWF-fil. Om du till exempel ska importera ett bibliotek med namnet *myClasses.swf* som finns i underkatalogen `lib` i programmets rotmapp, inkluderar du följande skripttagg i en HTML-fil:

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

Viktigt! *type-attributet måste vara* `type="application/x-shockwave-flash"` *för att biblioteket ska kunna läsas in på rätt sätt.*

Om SWF-innehållet har kompilerats som en SWF-fil för Flash Player 10 eller AIR 1.5 måste du ange XML-namnutrymmet i programbeskrivningsfilen till namnutrymmet för AIR 1.5.

`lib`-katalogen och `myClasses.swf`-filen måste också ingå när AIR-filen paketeras.

Få tillgång till de importerade klasserna via egenskapen `runtime` för JavaScript Window-objektet:

```
var libraryObject = new window.runtime.LibraryClass();
```

Om klasserna i SWF-filen är ordnade i paket måste du även ta med paketnamnet. Om LibraryClass-definitionen till exempel finns i ett paket med namnet *utilities* kan du skapa en instans av klassen med följande programsats:

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

Obs! *Om du vill kompilera ett ActionScript SWF-bibliotek så att det kan användas som en del av en HTML-sida i AIR, använder du* `acompc`*-kompilatorn. Verktyget acompc ingår i Flex SDK och beskrivs i dokumentationen för Flex SDK.*

Få tillgång till HTML DOM och JavaScript-objekt från en importerad ActionScript-fil

Adobe AIR 1.0 och senare

Om du vill få tillgång till objekt på en HTML-sida från ActionScript i en SWF-fil som är importerad till sidan med hjälp av `<script>`-taggen, skickar du en referens till ett JavaScript-objekt, t.ex. `window` eller `document`, till en funktion som definieras i ActionScript-koden. Använd referensen i funktionen för att få åtkomst till JavaScript-objektet (eller andra objekt som är tillgängliga via den skickade referensen).

Ta till exempel följande HTML-sida:

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

På den här enkla HTML-sidan finns en JavaScript-variabel som heter *num* och en JavaScript-funktion som heter *getStatus()*. De här båda är egenskaper för sidans `window`-objekt. `window.document`-objektet innehåller också ett namngivet P-element (med ID *p1*).

Sidan läser in en ActionScript-fil, ”ASLibrary.swf”, som innehåller en klass (ASClass). ASClass definierar en funktion med namnet `accessDOM()` som spårar värdena för dessa JavaScript-objekt. Metoden `accessDOM()` använder JavaScript Window-objektet som ett argument. Med hjälp av den här Window-referensen kan den få åtkomst till andra objekt på sidan, inklusive variabler, funktioner och DOM-element. Se följande definition:

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

Du kan både hämta och ange egenskaper för HTML-sidan från en importerad ActionScript-klass. Följande funktion anger till exempel innehållet för `p1`-elementet på sidan och den anger värdet för JavaScript-variabeln `foo` som finns på sidan:

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

Konvertera Date- och RegExp-objekt

Adobe AIR 1.0 och senare

Både JavaScript- och ActionScript-språken definierar klasserna Date och RegExp, men objekt av de här typerna konverteras inte automatiskt mellan de två körningskontexterna. Du måste konvertera Date- och RegExp-objekt till motsvarande typ innan du använder dem för att ange egenskaper eller funktionsparametrar i en annan körningskontext.

Följande ActionScript-kod konverterar till exempel ett Date-objekt i JavaScript med namnet `jsDate` till ett Date-objekt i ActionScript:

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

Följande ActionScript-kod konverterar ett RegExp-objekt i JavaScript med namnet `jsRegExp` till ett RegExp-objekt i ActionScript:

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

Ändra en HTML-formatmall från ActionScript

Adobe AIR 1.0 och senare

När HTMLLoader-objektet har skickat `complete`-händelsen kan du undersöka och ändra CSS-formaten på en sida.

Ta till exempel följande enkla HTML-dokument:

```
<html>
<style>
    .style1A { font-family:Arial; font-size:12px }
    .style1B { font-family:Arial; font-size:24px }
</style>
<style>
    .style2 { font-family:Arial; font-size:12px }
</style>
<body>
    <p class="style1A">
        Style 1A
    </p>
    <p class="style1B">
        Style 1B
    </p>
    <p class="style2">
        Style 2
    </p>
</body>
</html>
```

Efter att ett HTMLLoader-objekt har läst in sitt innehåll kan du förändra CSS-formaten på sidan via `cssRules`-arrayen för `window.document.styleSheets`-arrayen, vilket visas här:

```
 var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event):void {
    var styleSheet0:Object = html.window.document.styleSheets[0];
    styleSheet0.cssRules[0].style.fontSize = "32px";
    styleSheet0.cssRules[1].style.color = "#FF0000";
    var styleSheet1:Object = html.window.document.styleSheets[1];
    styleSheet1.cssRules[0].style.color = "blue";
    styleSheet1.cssRules[0].style.font-family = "Monaco";
}
```

Den här koden justerar CSS-formaten så att det slutliga HTML-dokumentet visas på följande sätt:

Style 1A

Style 1B

Style 2

Kom ihåg att koden kan lägga till format till sidan efter att HTMLLoader-objektet har skickat `complete`-händelsen.

Korsskriptning av innehåll i olika säkerhetssandlådor

Adobe AIR 1.0 och senare

Runtime-modulens säkerhetsmodell isolerar kod av olika ursprung. Genom att använda korsskriptning av innehåll i olika säkerhetssandlådor kan du tillåta att innehåll i en säkerhetssandlåda får tillgång till valda egenskaper och metoder i en annan sandlåda.

AIR-säkerhetssandlådor och JavaScript-kod

Adobe AIR 1.0 och senare

AIR använder en policy för samma ursprung som förhindrar att kod från en domän interagerar med innehåll i en annan. Alla filer placeras i en sandlåda utifrån sitt ursprung. I vanliga fall kan inte innehåll i programsandlådan bryta mot principen om samma ursprung och göra korsskriptning av innehåll som läses in från en annan plats utanför programinstallationskatalogen. I AIR finns dock ett par tekniker som gör att du kan utföra korsskriptning på icke-programinnehåll.

I en teknik används frames eller iframes för att mappa programinnehåll till en annan säkerhetssandlåda. Sidor som blir inlästa från programmets sandlådeområde fungerar på samma sätt som om de vore inlästa från en fjärrdomän. Om programinnehåll till exempel mappas till domänen *example.com* kan innehållet utföra korsskriptning på sidor som lästs in från example.com.

Eftersom den här tekniken placerar programinnehållet i en annan sandlåda, omfattas inte längre koden i innehållet av begränsningarna för körning av kod i utvärderade strängar. Du kan använda den här tekniken för sandlådemappning för att förenkla dessa begränsningar även om du inte behöver utföra korsskriptning på fjärrinnehåll. Att mappa innehåll på det här sättet kan vara användbart om du arbetar med ett av många JavaScript-ramverk eller med befintliga koder som är beroende av utvärderande strängar. Du bör dock överväga de risker det innebär om otillförlitligt innehåll infogas och utförs när innehåll körs utanför programsandlådan.

Programinnehåll som mappas till en annan sandlåda förlorar också åtkomsten till programgränssnitten i AIR, så tekniken för sandlådemappning kan inte användas för att göra AIR-funktioner tillgängliga för kod som körs utanför programsandlådan.

Med en annan teknik för korsskriptning kan du skapa ett gränssnitt, som kallas *sandlådebrygga*, mellan innehållet i en icke-programsandlåda och dess överordnade dokument i programsandlådan. Bryggan gör att det underordnade innehållet får åtkomst till egenskaper och metoder som har definierats av det överordnade innehållet, det överordnade innehållet får åtkomst till de egenskaper och metoder som definierats av det underordnade innehållet eller både och.

Du kan också utföra XMLHttpRequests mellan olika domäner från programsandlådan och från andra sandlådor.

Mer information finns i ”Bildrute- och iframe-element i HTML” på sidan 976, ”HTML-säkerhet i Adobe AIR” på sidan 1085 och ”Objektet XMLHttpRequest” på sidan 970.

Läsa in programinnehåll i en icke-programsandlåda

Adobe AIR 1.0 och senare

Om du vill tillåta att programinnehåll ska kunna utföra säker korsskriptning på innehåll som lästs in från en annan plats än programinstallationskatalogen, kan du använda elementen `frame` eller `iframe` för att läsa in programinnehåll i samma säkerhetssandlåda som det externa innehållet. Om du inte behöver utföra korsskriptning på fjärrinnehåll men ändå vill läsa in en sida av programmet som är utanför programsandlådan, kan du använda samma teknik och ange `http://localhost/` eller något annat neutralt värde som ursprungsdomän.

AIR lägger till nya attribut, `sandboxRoot` och `documentRoot`, till det frame-element som gör att du kan ange om en programfil som är inläst i ramen ska mappas till en icke-programsandlåda. Filer som relaterar till en sökväg under `sandboxRoot`-URL:en blir i stället inlästa från `documentRoot`-katalogen. Av säkerhetsskäl behandlas programinnehåll som lästs in på det här sättet som om det var inläst från `sandboxRoot`-URL:en.

Egenskapen `sandboxRoot` anger den URL som ska användas för att bestämma i vilken sandlåda och domän som raminnehållet ska placeras. URL-schemat `file:`, `http:` eller `https:` måste användas. Om du anger en relativ URL ligger innehållet kvar i programsandlådan.

Egenskapen `documentRoot` anger den katalog som raminnehållet ska läsas in från. URL-schemat `file:`, `app:` eller `app-storage:` måste användas.

I följande exempel mappas innehåll som är installerat i `sandbox`-underkatalogen i programmet så att det körs i fjärrsandlådan och domänen `www.example.com`:

```
 <iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

Sidan `ui.html` kunde läsa in en javascript-fil från den lokala `sandbox`-mappen med hjälp av följande skripttagg:

```
<script src="http://www.example.com/local/ui.js"></script>
```

Den kunde också läsa in innehåll från en katalog på fjärrservern med hjälp av en script-tagg som den nedan:

```
<script src="http://www.example.com/remote/remote.js"></script>
```

`sandboxRoot`-URL:en maskerar innehåll som finns på samma URL på fjärrservern. I exemplet ovan skulle du inte kunna få åtkomst till fjärrinnehåll på `www.example.com/local/` (eller någon av dess underkataloger), eftersom AIR mappar om begäran till den lokala programkatalogen. Begäran mappas om oavsett om den härleds från sidnavigering, från en XMLHttpRequest eller från något annat sätt att läsa in innehåll.

Konfigurera ett gränssnitt för en sandlådebrygga

Adobe AIR 1.0 och senare

Du kan använda en sandlådebrygga om innehåll i programsandlådan måste ha tillgång till egenskaper och metoder som definierats av innehåll i en icke-programsandlåda eller om icke-programinnehåll måste ha tillgång till egenskaper och metoder som definierats av innehåll i programsandlådan. Skapa en brygga med hjälp av egenskaperna `childSandboxBridge` och `parentSandboxBridge` i `window`-objektet för ett underordnat dokument.

Skapa en underordnad sandlådebrygga

Adobe AIR 1.0 och senare

Egenskapen `childSandboxBridge` gör att det underordnade dokumentet kan visa ett gränssnitt för innehåll i det överordnade dokumentet. Om du vill visa ett gränssnitt anger du egenskapen `childSandbox` som en funktion eller ett objekt i det underordnade dokumentet. Du kan sedan få åtkomst till objektet eller funktionen via innehåll i det överordnade dokumentet. Följande exempel visar hur ett skript som körs i ett underordnat dokument kan visa ett objekt som innehåller en funktion och en egenskap för dess överordnade dokument:

```
 var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Om det här underordnade innehållet lästes in i ett iframe-element som tilldelats ID ”child”, kan du få åtkomst till gränssnittet via det överordnade innehållet genom att läsa egenskapen `childSandboxBridge` i frame-elementet:

```
 var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

Skapa en överordnad sandlådebrygga

Adobe AIR 1.0 och senare

Egenskapen `parentSandboxBridge` gör att det överordnade dokumentet kan visa ett gränssnitt för innehåll i det underordnade dokumentet. Om du vill visa ett gränssnitt anger det överordnade dokumentet egenskapen `parentSandbox` i det underordnade dokumentet som en funktion eller ett objekt som definieras i det överordnade dokumentet. Du kan sedan få åtkomst till objektet eller funktionen via innehåll i det underordnade dokumentet. Följande exempel visar hur ett skript som körs i ett överordnat frame-element kan visa ett objekt som innehåller en funktion för ett underordnat dokument:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

Om du använder det här gränssnittet kan innehåll i det underordnade frame-elementet spara text till en fil med namnet `save.txt`, men det har ingen åtkomst till filsystemet. Det underordnade innehållet kan anropa save-funktionen på följande sätt:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

Programinnehåll ska visa ett så begränsat gränssnitt som möjligt för andra sandlådor. Icke-programinnehåll ska hanteras som otillförlitligt eftersom det kan vara utsatt för skadlig kod eller kod som angetts av misstag. Du måste vidta säkerhetsåtgärder för att förhindra obehörig användning av gränssnittet som du visar via den överordnade sandlådebryggan.

Åtkomst till överordnad sandlådebrygga under inläsning av sida

Adobe AIR 1.0 och senare

För att ett skript i ett underordnat dokument ska kunna få åtkomst till en överordnad sandlådebrygga måste bryggan vara inställd innan skriptet körs. Window-, frame- och iframe-objekten skickar en `dominitialize`-händelse när DOM för en ny sida har skapats, innan några skript har tolkats eller några DOM-element har lagts till. Du kan använda `dominitialize`-händelsen för att skapa bryggan tidigt i sidkonstruktionsprocessen så att alla skript i det underordnade dokumentet får åtkomst till den.

Följande exempel visa hur du skapar en överordnad sandlådebrygga som svar på den `dominitialize`-händelse som skickas från det underordnade frame-objektet:

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

Följande `child.html`-dokument visar hur underordnat innehåll får åtkomst till den överordnade sandlådebryggan:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Om du vill avlyssna `dominitialize`-händelsen i ett underordnat window-objekt, i stället för ett frame-objekt, måste du lägga till avlyssnaren i det nya underordnade window-objektet som skapas av `window.open()`-funktionen:

```
var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

I så fall går det inte att mappa programinnehåll till en icke-programsandlåda. Den här tekniken är bara användbar när `child.html` har lästs in från en plats utanför programkatalogen. Du kan fortfarande mappa programinnehåll i window-objektet till en icke-programsandlåda, men du måste först läsa in en mellanliggande sida som själv använder ramar för att läsa in det underordnade dokumentet och mappa det till önskad sandlåda.

Om du använder funktionen `createRootWindow()` i klassen HTMLLoader för att skapa ett fönster är det nya fönstret inte underordnat det dokument som `createRootWindow()` anropas från. Du kan därför inte skapa en sandlådebrygga från det anropande fönstret till icke-programinnehåll som blir inläst i det nya fönstret. Du måste i stället läsa in en mellanliggande sida i det nya fönstret som själv använder ramar för att kunna läsa in det underordnade dokumentet. Du kan sedan skapa bryggan från det överordnade dokumentet i det nya fönstret till det underordnade dokumentet som blir inläst i ramen.

Kapitel 60: Skripta AIR HTML-behållaren

Adobe AIR 1.0 och senare

Klassen HTMLLoader fungerar som behållare för HTML-innehåll i Adobe® AIR®. Klassen innehåller många egenskaper och metoder, som ärvts från klassen Sprite, som du kan använda för att kontrollera beteendet och utseendet för objekt i ActionScript® 3.0-visningslistan. Dessutom definierar klassen egenskaper och metoder för uppgifter som att läsa in och interagera med HTML-innehåll och hantera historik.

Klassen HTMLHost används för att definiera en uppsättning standardbeteenden för en HTMLLoader. När du skapar ett HTMLLoader-objekt, innehåller det ingen HTMLHost-implementering. Detta medför att ingenting händer när HTML-innehåll triggar ett standardbeteende som att till exempel byta fönsterplacering eller fönstertitel. Du kan utöka klassen HTMLHost till att definiera de beteenden som du vill ha för ditt program.

HTML-fönster som skapas av AIR innehåller en standardimplementering av HTMLHost. Du kan tilldela standardimplementeringen av HTMLHost till ett annat HTMLLoader-objekt genom att ställa in egenskapen `htmlHost` i objektet med hjälp av ett nytt HTMLHost-objekt som skapas med parametern `defaultBehavior` inställd till `true`.

Obs! *I Adobe® Flex™ Framework kapslas objektet HTMLLoader in av komponenten mx:HTML. Använd HTML-komponenten när du använder Flex.*

Visningsegenskaper för HTMLLoader-objekt

Adobe AIR 1.0 och senare

Ett HTMLLoader-objekt ärver visningsegenskaperna från klassen Sprite i Adobe® Flash® Player. Du kan till exempel ändra storlek, flytta och dölja objektet eller ändra bakgrundsfärgen. Du kan också tillämpa avancerade effekter som filter, masker, skalning eller rotation. När du använder effekter bör du tänka på hur de påverkar läsbarheten. SWF- och PDF-innehåll som läses in på en HTML-sida kan inte visas när vissa effekter används.

HTML-fönster innehåller ett HTMLLoader-objekt som renderar HTML-innehållet. Detta objekt begränsas av fönstrets yta, vilket gör att ändringar av dimensioner, position, rotation eller skalfaktor inte alltid ger önskvärda resultat.

Grundläggande visningsegenskaper

Adobe AIR 1.0 och senare

De grundläggande visningsegenskaperna i HTMLLoader ger dig möjlighet att placera kontrollen i dess överordnade visningsobjekt, att ange storleken och att visa eller dölja kontrollen. Du bör inte ändra dessa egenskaper för HTMLLoader-objektet för ett HTML-fönster.

De grundläggande egenskaperna innefattar:

Egenskap	Anteckningar
`x`, `y`	Placerar objektet inuti dess överordnade behållare.
`width`, `height`	Ändrar visningsytans dimensioner.
`visible`	Kontrollerar synligheten för objektet och allt innehåll i objektet.

Utanför ett HTML-fönster är standardinställningen för egenskaperna `width` och `height` satt till 0. Du måste ställa in bredd och höjd innan det går att se det inlästa HTML-innehållet. HTML-innehåll ritas efter storleken på HTMLLoader, och layoutas enligt HTML- och CSS-egenskaperna i innehållet. När storleken på HTMLLoader ändras, flödas innehållet om.

När du läser in innehåll i ett nytt HTMLLoader-objekt (med `width` fortfarande satt till 0), kan det vara frestande att ställa in `width` och `height` för visning av HTMLLoader-objektet med egenskaperna `contentWidth` och `contentHeight`. Denna teknik fungerar för sidor som har en rimlig minimibredd när de layoutas enligt flödesreglerna i HTML och CSS. Vissa sidor flödar dock i en lång och smal layout när en rimlig bredd inte anges i HTMLLoader-objektet.

Obs! *När du ändrar bredd och höjd på ett HTMLLoader-objekt ändras inte värdena för scaleX och scaleY, vilket skulle ske med de flesta andra typer av visningsobjekt.*

Genomskinlighet för innehåll i HTMLLoader

Adobe AIR 1.0 och senare

Egenskapen `paintsDefaultBackground` för ett HTMLLoader-objekt är som standard inställt till `true`, och avgör om HTMLLoader-objektet ritar en ogenomskinlig bakgrund. När `paintsDefaultBackground` är `false`, är bakgrunden genomskinlig. Bakom förgrundselementen i HTML-innehållet, går det att se behållaren för visningsobjektet eller andra visningsobjekt som ligger under HTMLLoader-objektet.

Om body-elementet eller något annat element i HTML-dokumentet specificerar en bakgrundsfärg (med till exempel `style="background-color:gray"`), är bakgrunden för den delen av HTML-koden ogenomskinlig, och renderas med den angivna bakgrundsfärgen. Om du ställer in egenskapen `opaqueBackground` för HTMLLoader-objektet och `paintsDefaultBackground` är satt till `false`, är färgen som är inställd för `opaqueBackground` synlig.

Obs! *Du kan använda en genomskinlig bild i PNG-format för att skapa en alfablandad bakgrund för ett element i ett HTML-dokument. Att ställa in opacitetstyp för ett HTML-element stöds inte.*

Skala HTMLLoader-innehåll

Adobe AIR 1.0 och senare

Undvik att skala ett HTMLLoader-objekt över en skalfaktor på 1,0. Text i HTMLLoader-innehåll renderas med en specifik upplösning och ser taggig ut om HTMLLoader-objektet skalas upp. Ställ in scenegenskapen `scaleMode` till `StageScaleMode.NO_SCALE` för att förhindra att HTMLLoader och allt dess innehåll skalas när storleken på fönstret ändras.

Att tänka på vid inläsning av SWF- eller PDF-innehåll i en HTML-sida

Adobe AIR 1.0 och senare

SWF- och PDF-innehåll som läses in i ett HTMLLoader-objekt försvinner under följande förhållanden:

- Om du skalar HTMLLoader-objektet till en annan faktor än 1,0.
- Om du ställer in egenskapen alpha i HTMLLoader-objektet till ett annat värde än 1,0.
- Om du roterar HTMLLoader-innehållet.

Innehållet visas igen om du tar bort egenskapsinställningen som orsakade problemet och tar bort de aktiva filtren.

Dessutom kan inte körningstiden visa PDF-innehållet i genomskinliga fönster. Körningstiden visar endast SWF-innehåll som är inbäddat i en HTML-sida när parametern `wmode` för objektet eller den inbäddade taggen är inställd på `opaque` eller `transparent`. Eftersom standardvärdet för `wmode` är `window` visas inte SWF-innehåll i genomskinliga fönster såvida du inte uttryckligen anger wmode-parametern.

Obs! *Före AIR 1.5.2 kunde inte SWF som är inbäddat i HTML visas, oavsett vilket wmode-värde som användes.*

Mer information om hur du läser in den här typen av media i ett HTMLLoader-objekt finns i avsnitten ”Bädda in SWF-innehåll i HTML” på sidan 997 och ”Lägga till PDF-innehåll i AIR” på sidan 551.

Avancerade visningsegenskaper

Adobe AIR 1.0 och senare

Klassen HTMLLoader ärver flera metoder som kan användas som specialeffekter. För det mesta har dessa effekter begränsningar när de används med HTMLLoader-visningen, med de kan vara användbara till övergångar eller andra tillfälliga effekter. Om du till exempel visar ett dialogfönster för att hämta inmatningar från en användare, kan du göra visningen av huvudfönstret suddig till dess att användaren stänger dialogen. På samma sätt kan du tona ut visningen när du stänger ett fönster.

De avancerade visningsegenskaperna innefattar:

Egenskap	Begränsningar
`alpha`	Kan minska läsbarheten på HTML-innehåll
`filters`	Yttre effekter i ett HTML-fönster klipps bort av fönstrets kant
`graphics`	Former som är ritade med graphics-kommandon visas under HTML-innehåll, inklusive standardbakgrunden. Egenskapen paintsDefaultBackground måste vara satt till false för att de ritade formerna ska synas.
`opaqueBackground`	Ändrar inte färgen på standardbakgrunden. Egenskapen paintsDefaultBackground måste vara satt till false för att detta färglager ska synas.
`rotation`	Hörnen på det rektangulära HTMLLoader-området kan klippas bort av fönstrets kant. SWF- och PDF-innehåll som är inläst i HTML-innehållet visas inte.
`scaleX`, `scaleY`	Den renderade visningen kan se taggig ut vid skalfaktorer större än 1. SWF- och PDF-innehåll som är inläst i HTML-innehållet visas inte.
`transform`	Kan minska läsbarheten på HTML-innehåll. HTML-visningen kan klippas bort av fönstrets kant. SWF- och PDF-innehåll som är inläst i HTML-innehållet visas inte om omformningen innefattar rotation, skalning eller skevning.

Följande exempel visar hur du ska ställa in arrayen `filters` för att göra hela HTML-visningen suddig:

```
 var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
html.width = 800;
html.height = 600;

var blur:BlurFilter = new BlurFilter(8);
var filters:Array = [blur];
html.filters = filters;
```

Rulla HTML-innehåll

Adobe AIR 1.0 och senare

Klassen HTMLLoader inkluderar följande egenskaper för att kontrollera rullningen av HTML-innehåll:

Egenskap	Beskrivning
`contentHeight`	Höjden i pixlar för HTML-innehållet.
contentWidth	Bredden i pixlar för HTML-innehållet.
scrollH	Den vågräta rullningspositionen för HTML-innehållet i HTMLLoader-objektet.
scrollV	Den lodräta rullningspositionen för HTML-innehållet i HTMLLoader-objektet.

Följande kod ställer in egenskapen `scrollV` så att HTML-innehåll rullas till nederkanten av sidan:

```
 var html:HTMLLoader = new HTMLLoader();
html.addEventListener(Event.HTML_BOUNDS_CHANGE, scrollHTML);

const SIZE:Number = 600;
html.width = SIZE;
html.height = SIZE;

var urlReq:URLRequest = new URLRequest("http://www.adobe.com");
html.load(urlReq);
this.addChild(html);

function scrollHTML(event:Event):void
{
    html.scrollV = html.contentHeight - SIZE;
}
```

HTMLLoader innehåller inte vågräta och lodräta rullningslister. Du kan implementera rullningslister i ActionScript eller genom att använda en Flex-komponent. HTML-Flex-komponenten inkluderar automatiskt rullningslister för HTML-innehåll. Du kan också använda metoden `HTMLLoader.createRootWindow()` för att skapa ett fönster som innehåller ett HTMLLoader-objekt med rullningslister (se ”Skapa fönster med rullande HTML-innehåll” på sidan 1023).

Öppna HTML-historiklistan

Adobe AIR 1.0 och senare

När nya sidor läses in i ett HTMLLoader-objekt, behåller körtiden en historiklista för objektet. Historiklistan motsvarar HTML-sidans `window.history`-objekt. Klassen HTMLLoader innehåller följande egenskaper och metoder för att arbeta med HTML-historiklistan:

Klassmedlem	Beskrivning
`historyLength`	Den sammanlagda längden på historiklistan, inklusive poster för bakåt- och framåtnavigering.
`historyPosition`	Den aktuella positionen i historiklistan. Historikobjekt före denna position representerar bakåtnavigering. Objekt efter positionen representerar framåtnavigering.
`getHistoryAt()`	Returnerar det URLRequest-objekt som motsvarar historikposten vid den angivna positionen i historiklistan.
`historyBack()`	Navigerar bakåt i historiklistan, om det är möjligt.
`historyForward()`	Navigerar framåt i historiklistan, om det är möjligt.
`historyGo()`	Navigerar angivet antal steg i webbläsarhistoriken. Navigerar framåt om värdet är positivt och bakåt om värdet är negativt. Navigering till noll läser in sidan igen. Att ange en position efter slutet navigerar till slutet på listan.

Objekt i historiklistan sparas som objekt med typen HTMLHistoryItem. Klassen HTMLHistoryItem har följande egenskaper:

Egenskap	Beskrivning
`isPost`	Ställd till `true` om HTML-sidan innehåller POST-data.
`originalUrl`	HTML-sidans ursprungliga URL, före eventuella omdirigeringar.
`title`	HTML-sidans titel.
`url`	HTML-sidans URL.

Ställa in den användaragent som används vid inläsning av HTML-innehåll

Adobe AIR 1.0 och senare

Klassen HTMLLoader har en egenskap, `userAgent`, med vilken du kan ange den användaragentsträng som ska användas av HTMLLoader. Ställ in egenskapen `userAgent` för HTMLLoader-objektet innan du anropar metoden `load()`. Egenskapen `userAgent` för den URLRequest som skickas till metoden `load()` används *inte* om du ställer in egenskapen i HTMLLoader-instansen.

Du kan ange den standardinställda användaragentsträng som ska användas av alla HTMLLoader-objekt i en programdomän genom att ange egenskapen `URLRequestDefaults.userAgent`. De statiska egenskaperna URLRequestDefaults används som standard för alla URLRequest-objekt, och inte bara för de URLRequest som används med metoden `load()` för HTMLLoader-objekt. Om egenskapen `userAgent` anges för HTMLLoader-objektet åsidosätts inställningen `URLRequestDefaults.userAgent`.

Om du inte anger ett användaragentvärde för antingen egenskapen `userAgent` för HTMLLoader-objektet, eller för `URLRequestDefaults.userAgent`, används standardvärdet för användaragent i AIR. Det här standardvärdet varierar beroende på körtidens operativsystem (till exempel Mac OS eller Windows), körtidens språk och körtidens version, som i följande två exempel:

- `"Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, t.ex. Gecko) AdobeAIR/1.0"`
- `"Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, t.ex. Gecko) AdobeAIR/1.0"`

Ställa in teckenkodning att använda för HTML-innehåll

Adobe AIR 1.0 och senare

På en HTML-sida kan teckenkodningen som används specificeras genom inkludering av en `meta`-tagg, som till exempel följande:

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

Åsidosätt sidinställningen för att säkerställa att en specifik teckenkodning används genom att ställa in egenskapen `textEncodingOverride` för HTMLLoader-objektet:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingOverride = "ISO-8859-1";
```

Specificera den teckenkodning som ska användas för HTMLLoader-innehåll när en HTML-sida inte specificerar en inställning med egenskapen `textEncodingFallback` för HTMLLoader-objektet:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingFallback = "ISO-8859-1";
```

Egenskapen `textEncodingOverride` åsidosätter inställningen i HTML-sidan. Egenskapen `textEncodingOverride` och inställningen i HTML-sidan åsidosätter egenskapen `textEncodingFallback`.

Ställ in egenskapen `textEncodingOverride` eller egenskapen `textEncodingFallback` innan du läser in HTML-innehållet.

Definiera webbläsarliknande användargränssnitt för HTML-innehåll

Adobe AIR 1.0 och senare

JavaScript innehåller flera API:er för att kontrollera det fönster som visar HTML-innehållet. I AIR kan dessa API:er skrivas över genom att implementera en anpassad HTMLHost-klass.

Om att utöka klassen HTMLHost

Adobe AIR 1.0 och senare

Om ditt program till exempel visar flera HTMLLoader-objekt i ett flikgränssnitt, kan du önska att titeländringar som görs av de inlästa HTML-sidorna ska ändra flikens titel, och inte huvudfönstrets titel. På samma sätt skulle koden kunna svara på ett `window.moveTo()`-anrop genom att placera om HTMLLoader-objektet i dess överordnade visningsobjektbehållare, genom att flytta fönstret som innehåller HTMLLoader-objektet, genom att inte göra något eller genom att göra något helt annat.

Klassen HTMLHost i AIR används för att kontrollera följande JavaScript-egenskaper och -metoder:

- `window.status`
- `window.document.title`
- `window.location`
- `window.blur()`
- `window.close()`
- `window.focus()`
- `window.moveBy()`
- `window.moveTo()`
- `window.open()`
- `window.resizeBy()`
- `window.resizeTo()`

När du skapar ett HTMLLoader-objekt med `new HTMLLoader()`, aktiveras inte de listade JavaScript-egenskaperna och -modellerna. Klassen HTMLHost innehåller en webbläsarliknande standardimplementering av dessa JavaScript API:er. Du kan också utöka klassen HTMLHost för att anpassa beteendet. Skapa ett HTMLHost-objekt som stöder standardbeteendet genom att ställa parametern `defaultBehaviors` till true i HTMLHost-konstruktorn.

```
var defaultHost:HTMLHost = new HTMLHost(true);
```

När du skapar ett HTML-fönster i AIR med metoden `createRootWindow()` från klassen HTMLLoader, tilldelas automatiskt en HTMLHost-instans som stöder standardbeteendena. Du kan ändra host-objektets beteende genom att tilldela en annan HTMLHost-implementering till egenskapen `htmlHost` i HTMLLoader, eller tilldela `null` för att inaktivera funktionerna helt.

Obs! *I AIR tilldelas ett standard-HTMLHost-objekt till det ursprungliga fönstret som skapades för ett HTML-baserat AIR-program, samt till alla fönster som skapas av standardimplementeringen av JavaScript-metoden* `window.open()`.

Exempel: Utöka klassen HTMLHost

Adobe AIR 1.0 och senare

Följande exempel visar hur du ska anpassa det sätt som ett HTMLLoader-objekt påverkar användargränssnittet genom att utöka klassen HTMLHost:

Flex-exempel:

1 Skapa en klass som utökar klassen HTMLHost (en underklass).

2 Åsidosättningsmetoder i den nya klassen för att hantera ändringar i de användargränssnittsrelaterade inställningarna. Till exempel definierar följande klass, CustomHost, beteenden för anrop av `window.open()` och ändringar av `window.document.title`. Anrop av `window.open()` öppnar HTML-sidan i ett nytt fönster. Ändringar av `window.document.title` (inklusive inställning av elementet `<title>` i en HTML-sida) ställer in titeln för det fönstret.

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                                 windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

3 Skapa ett objekt av den nya klassen i koden som innehåller HTMLLoader-objektet (inte koden för den nya underklassen till HTMLHost). Tilldela det nya objektet till egenskapen `htmlHost` för HTMLLoader-objektet. Följande Flex-kod använder klassen CustomHost som definierades i det föregående steget:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

Testa den kod som beskrivs här genom att inkludera en HTML-fil med följande innehåll i programkatalogen:

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

Flash Professional-exempel:

1 Skapa en Flash-fil för AIR. Ställ in dess dokumentklass till CustomHostExample och spara filen som CustomHostExample.fla.

2 Skapa en ActionScript-fil med namnet CustomHost.as, som innehåller en klass som utökar klassen HTMLHost (en underklass). Klassen åsidosätter vissa metoder i den nya klassen för att hantera ändringar i de användargränssnittsrelaterade inställningarna. Till exempel definierar följande klass, CustomHost, beteenden för anrop av `window.open()` och ändringar av `window.document.title`. Anrop av metoden `window.open()` öppnar HTML-sidan i ett nytt fönster. Ändringar av egenskapen `window.document.title` (inklusive intställning av elementet `<title>` i en HTML-sida) ställer in titeln för det fönstret.

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                        windowCreateOptions.y,
                                        windowCreateOptions.width,
                                        windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                    windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
```

```
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

3 Skapa en annan ActionScript-fil med namnet CustomHostExample.as, som innehåller dokumentklassen för programmet. Denna klass skapar ett HTMLLoader-objekt och ställer in dess host-egenskap till en instans av klassen CustomHost som definierades i det föregående steget:

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

Testa den kod som beskrivs här genom att inkludera en HTML-fil med följande innehåll i programkatalogen:

```
<html>
    <head>
    <title>Test</title>
    <script>
    function openWindow()
    {
    document.title = "Test"
    window.open('Test.html');
    }
    </script>
    </head>
    <body bgColor="#EEEEEE">
    <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
    <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
    window.document.location = 'http://www.adobe.com'</a>
    <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
    <br/><a href="#" onclick="window.close()">window.close()</a>
    <br/><a href="#" onclick="window.blur()">window.blur()</a>
    <br/><a href="#" onclick="window.focus()">window.focus()</a>
    <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
    </body>
</html>
```

Hantera ändringar av egenskapen window.location

Adobe AIR 1.0 och senare

Åsidosätt metoden `locationChange()` för att hantera ändringar av HTML-sidans URL. Metoden `locationChange()` anropas när JavaScript på en sida ändrar värdet för `window.location`. Följande exempel läser in den begärda URL:en:

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

Obs! *Du kan använda egenskapen htmlLoader i HTMLHost-objektet för att referera det aktuella HTMLLoader-objektet.*

Hantera JavaScript-anrop till window.moveBy(), window.moveTo(), window.resizeTo(), window.resizeBy()

Adobe AIR 1.0 och senare

Åsidosätt metoden `set windowRect()` för att hantera ändringar av HTML-innehållets gränser. Metoden `set windowRect()` anropas när JavaScript på en sida anropar `window.moveBy()`, `window.moveTo()`, `window.resizeTo()` eller `window.resizeBy()`. Följande exempel uppdaterar gränserna för skrivbordsfönstret:

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

Hantera JavaScript-anrop till window.open()

Adobe AIR 1.0 och senare

Åsidosätt metoden `createWindow()` för att hantera JavaScript-anrop till `window.open()`. Implementeringar av metoden `createWindow()` används för att skapa och returnera ett nytt HTMLLoader-objekt. Vanligtvis visar du HTMLLoader i ett nytt fönster, men det är inte ett krav att skapa ett fönster.

Följande exempel visar hur du ska implementera funktionen `createWindow()` med `HTMLLoader.createRootWindow()` för att skapa både fönstret och HTMLLoader-objektet. Du kan också skapa ett NativeWindow-objekt separat, och lägga till HTMLLoader till fönsterscenen.

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                 windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

Obs! *Detta exempel tilldelar den anpassade implementeringen av HTMLHost till alla nya fönster som skapas med `window.open()`. Du kan också använda en annan implementering eller ställa in egenskapen `htmlHost` till null för nya fönster, om du så önskar.*

Objektet som skickas som en parameter till `createWindow()`-metoden är ett HTMLWindowCreateOptions-objekt. Klassen HTMLWindowCreateOptions innehåller egenskaper som rapporterar de värden som ställs in i parametersträngen `features` i anropet till `window.open()`:

Egenskapen HTMLWindowCreateOptions	Motsvarande inställning i egenskapssträngen i JavaScript-anropet till window.open()
`fullscreen`	`fullscreen`
`height`	`height`
`locationBarVisible`	`location`
`menuBarVisible`	`menubar`
`resizeable`	`resizable`
`scrollBarsVisible`	`scrollbars`
`statusBarVisible`	`status`
`toolBarVisible`	`toolbar`
`width`	`width`
`x`	`left` eller `screenX`
`y`	`top` eller `screenY`

Klassen HTMLLoader implementerar inte alla funktioner som kan specificeras i funktionssträngen. Ditt program måste tillhandahålla rullningslister, platsfält, menyfält, statusfält och verktygsfält när det är lämpligt.

Övriga argument till JavaScript-metoden `window.open()` hanteras av systemet. En implementering av `createWindow()` ska inte läsa in innehåll i HTMLLoader-objektet, eller ställa in fönstertiteln.

Hantera JavaScript-anrop till window.close()

Adobe AIR 1.0 och senare

Åsidosätt `windowClose()` för att hantera JavaScript-anrop till metoden `window.close()`. Följande exempel stänger skrivbordsfönstret när metoden `window.close()` anropas:

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

JavaScript-anrop till `window.close()` måste inte stänga det innehållande fönstret. Du kan till exempel ta bort HTMLLoader från visningslistan och lämna fönstret (som kan ha annat innehåll) öppet, som i följande kod:

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

Hantera ändringar av egenskapen window.property

Adobe AIR 1.0 och senare

Åsidosätt metoden `updateStatus()` för att hantera JavaScript-ändringar av värdet för `window.status`. I följande exempel skrivs statusvärdet ut:

```
 override public function updateStatus(status:String):void
{
    trace(status);
}
```

Begärd status skickas som en sträng till metoden `updateStatus()`.

HTMLLoader-objektet innehåller inte ett statusfält.

Hantera ändringar av egenskapen window.document.title

Adobe AIR 1.0 och senare

Åsidosätt metoden `updateTitle()` för att hantera JavaScript-ändringar av värdet för `window.document.title`. Följande exempel ändrar fönstertiteln och lägger till strängen ”Prov” till titeln:

```
 override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

När `document.title` är inställt för en HTML-sida, skickas begärd titel som en sträng till metoden `updateTitle()`.

Ändringar av `document.title` måste inte ändra titeln på det fönster som innehåller HTMLLoader-objektet. Du kan exempelvis ändra ett annat gränssnittselement, som till exempel ett textfält.

Hantera JavaScript-anrop av window.blur() och window.focus()

Adobe AIR 1.0 och senare

Åsidosätt metoderna `windowBlur()` och `windowFocus()` för att hantera JavaScript-anrop till metoderna `window.blur()` och `window.focus()`, som det visas i följande exempel:

```
 override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

Obs! *AIR innehåller inte något API för att inaktivera ett fönster eller program.*

Skapa fönster med rullande HTML-innehåll

Adobe AIR 1.0 och senare

Klassen HTMLLoader innehåller en statisk metod, `HTMLLoader.createRootWindow()`, med vilken du kan öppna ett nytt fönster (som representeras av ett NativeWindow-objekt) som innehåller ett HTMLLoader-objekt och definierar några inställningar av användargränssnitt för det fönstret. Metoden tar fyra parametrar, med vilka du kan definiera användargränssnittet:

Parameter	Beskrivning
`visible`	Ett booleskt värde som anger om fönstret från början är synligt (`true`) eller inte (`false`).
`windowInitOptions`	Ett NativeWindowInitOptions-objekt. Klassen NativeWindowInitOptions definierar initieringsalternativ för ett NativeWindow-objekt, inklusive följande: Huruvida fönstret är minimerbart, maxmimerbart eller om det går att ändra storleken, huruvida systemet har systemfärg eller anpassad färg, huruvida fönstret är genomskinligt eller ej (för fönster som inte använder systemfärg), samt typen av fönster.
`scrollBarsVisible`	Om det finns rullningslister (`true`) eller ej (`false`).
`bounds`	Ett Rectangle-objekt som definierar placering och storlek på det nya fönstret.

Följande kod använder till exempel metoden `HTMLLoader.createRootWindow()` för att skapa ett fönster med HTMLLoader-innehåll som använder rullningslister:

```
 var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
var bounds:Rectangle = new Rectangle(10, 10, 600, 400);
var html2:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2:URLRequest = new URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

Obs! *Fönster som skapas genom att anropa `createRootWindow()` direkt i JavaScript, förblir oberoende av det öppnande HTML-fönstret. JavaScript-fönsteregenskaperna `opener` och `parent` är till exempel `null`. Om du däremot anropar `createRootWindow()` indirekt genom att åsidosätta HTMLHost-metoden `createWindow()` för att anropa `createRootWindow()`, refererar `opener` och `parent` till det öppnande HTML-fönstret.*

Skapa underklasser av klassen HTMLLoader

Adobe AIR 1.0 och senare

Du kan skapa en underklass av klassen HTMLLoader för att skapa nya beteenden. Du kan till exempel skapa en underklass som definierar standardhändelseavlyssnare för HTMLLoader-händelser (som exempelvis de händelser som skickas när HTML renderas eller när användaren klickar på en länk).

Följande exempel utökar klassen HTMLHost för att skapa *normal*-beteende när JavaScript-metoden `window.open()` anropas. Exemplet definierar därefter en underklass av HTMLLoader som använder den anpassade implementeringen av klassen HTMLHost:

```
package
{
       import flash.html.HTMLLoader;
    public class MyHTMLHost extends HTMLHost
       {
           public function MyHTMLHost()
           {
               super(false);
        }
        override public function createWindow(opts:HTMLWindowCreateOptions):void
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(opts.x, opts.y, opts.width, opts.height);
            var html:HTMLLoader = HTMLLoader.createRootWindow(true,
                                                    initOptions,
                                                    opts.scrollBarsVisible,
                                                    bounds);
            html.stage.nativeWindow.orderToFront();
            return html
       }
}
```

Följande definierar en underklass av klassen HTMLLoader som tilldelar ett MyHTMLHost-objekt till dess `htmlHost`-egenskap.

```
package
{
       import flash.html.HTMLLoader;
    import MyHTMLHost;
    import HTMLLoader;
       public class MyHTML extends HTMLLoader
       {
           public function MyHTML()
           {
               super();
               htmlHost = new MyHTMLHost();
           }
       }
}
```

Detaljer om klassen HTMLHost och metoden `HTMLLoader.createRootWindow()` som används i detta exempel finns under ”Definiera webbläsarliknande användargränssnitt för HTML-innehåll” på sidan 1013.

Kapitel 61: Hantera HTML-relaterade händelser i AIR

Adobe AIR 1.0 och senare

Med hjälp av ett händelsehanteringssystem kan programmerare besvara användarinmatningar och systemhändelser på ett smidigt sätt. Händelsemodellen i Adobe® AIR® är inte bara smidig och bekväm, utan uppfyller också gällande standarder. Den nya händelsemodellen bygger på DOM Level 3-händelsespecifikationen, som är en standardarkitektur för händelsehantering. Modellen erbjuder ett effektivt och samtidigt intuitivt händelsehanteringsverktyg för programmerare.

HTMLLoader-händelser

Adobe AIR 1.0 och senare

Ett HTMLLoader-objekt skickar följande Adobe® ActionScript® 3.0-händelser:

Händelse	Beskrivning
`htmlDOMInitialize`	Skickas när HTML-dokumentet skapas men innan några skript tolkas eller DOM-noder läggs till på sidan.
`complete`	Skickas när HTML DOM har skapats som svar på en inläsningsåtgärd, omedelbart efter `onload`-händelsen på HTML-sidan.
`htmlBoundsChanged`	Skickas när en av egenskaperna `contentWidth` och `contentHeight`, eller båda två, har ändrats.
`locationChange`	Skickas när location-egenskapen för HTMLLoader har ändrats.
`locationChanging`	Skickas innan platsen för HTMLLoader ändras på grund av användarnavigering, ett JavaScript-anrop eller en omdirigering. Händelsen `locationChanging` skickas inte när du anropar metoderna `load()`, `loadString()`, `reload()`, `historyGo()`, `historyForward()` eller `historyBack()`. Ett anrop av metoden `preventDefault()` för det skickade händelseobjektet avbryter navigeringen. Om en länk öppnas i systemwebbläsaren skickas inte en locationChanging-händelse eftersom HTMLLoader inte ändrar plats.
`scroll`	Skickas då HTML-motorn ändrar rullningspositionen. Scroll-händelser kan inträffa på grund av navigering till ankarlänkar (#-länkar) på sidan eller på grund av anrop till metoden `window.scrollTo()`. Att ange text i en textinmatning eller ett textområde kan också orsaka en scroll-händelse.
`uncaughtScriptException`	Skickas när ett JavaScript-undantag inträffar i HTMLLoader och undantaget inte fångas upp i JavaScript-koden.

Du kan också registrera en ActionScript-funktion för en JavaScript-händelse (t.ex. `onClick`). Mer information finns i ”Hantera DOM-händelser med ActionScript” på sidan 1026.

Hantera DOM-händelser med ActionScript

Adobe AIR 1.0 och senare

Du kan registrera ActionScript-funktioner som ska besvara JavaScript-händelser. Ta till exempel följande HTML-innehåll:

```
<html>
<body>
    <a href="#" id="testLink">Click me.</a>
</html>
```

Du kan registrera en ActionScript-funktion som en hanterare för alla händelser på sidan. Följande kod lägger till exempel till funktionen `clickHandler()` som avlyssnare för `onclick`-händelsen i `testLink`-elementet på HTML-sidan:

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event):void {
    html.window.document.getElementById("testLink").onclick = clickHandler;
}

function clickHandler( event:Object ):void {
    trace("Event of type: " + event.type );
}
```

Det händelseobjekt som skickas har inte typen flash.events.Event och är inte någon Event-underklass. Använd klassen Object för att deklarera en typ för händelsehanterarens funktionsargument.

Du kan också använda metoden `addEventListener()` för att registrera de här händelserna. Du kan till exempel ersätta metoden `completeHandler()` i det föregående exemplet med följande kod:

```
function completeHandler(event:Event):void {
    var testLink:Object = html.window.document.getElementById("testLink");
    testLink.addEventListener("click", clickHandler);
}
```

När en avlyssnare refererar till ett särskilt DOM-element kan det vara bra att vänta på att det överordnade HTMLLoader-objektet ska skicka `complete`-händelsen innan händelseavlyssnarna läggs till. HTML-sidor läser ofta in flera filer och HTML DOM är inte helt färdig förrän alla filerna är inlästa och tolkade. HTMLLoader skickar `complete`-händelsen när alla element har skapats.

Svara på JavaScript-undantag som inte fångats upp

Adobe AIR 1.0 och senare

Ta följande HTML:

```
<html>
<head>
    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

Det innehåller en JavaScript-funktion, `throwError()`, som refererar till en okänd variabel, `melbaToast`:

```
var x = 400 * melbaToast;
```

Om en JavaScript-åtgärd stöter på en ogiltig åtgärd som inte fångas upp i JavaScript-koden med en `try`/`catch`-struktur, skickar det HTMLLoader-objekt som innehåller sidan en HTMLUncaughtScriptExceptionEvent-händelse. Du kan registrera en hanterare för den här händelsen, se följande kod:

```
var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.width = container.width;
html.height = container.height;
container.addChild(html);
html.addEventListener(HTMLUncaughtScriptExceptionEvent.UNCAUGHT_SCRIPT_EXCEPTION,
                          htmlErrorHandler);
function htmlErrorHandler(event:HTMLUncaughtJavaScriptExceptionEvent):void
{
    event.preventDefault();
    trace("exceptionValue:", event.exceptionValue)
    for (var i:int = 0; i < event.stackTrace.length; i++)
    {
        trace("sourceURL:", event.stackTrace[i].sourceURL);
        trace("line:", event.stackTrace[i].line);
        trace("function:", event.stackTrace[i].functionName);
    }
}
```

I JavaScript kan du hantera samma händelse med hjälp av egenskapen window.htmlLoader:

```
<html>
<head>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>

    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }

        function htmlErrorHandler(event) {
            event.preventDefault();
            var message = "exceptionValue:" + event.exceptionValue + "\n";
            for (var i = 0; i < event.stackTrace.length; i++){
                message += "sourceURL:" + event.stackTrace[i].sourceURL +"\n";
                message += "line:" + event.stackTrace[i].line +"\n";
                message += "function:" + event.stackTrace[i].functionName + "\n";
            }
            alert(message);
        }

        window.htmlLoader.addEventListener("uncaughtScriptException", htmlErrorHandler);
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

`htmlErrorHandler()`-händelsehanteraren avbryter händelsens standardbeteende (som är att skicka JavaScript-felmeddelandet till AIR trace-utdata) och skickar ett eget utdatameddelande. Det returnerar värdet för `exceptionValue` för HTMLUncaughtScriptExceptionEvent-objektet. Det returnerar egenskaperna för varje objekt i `stackTrace`-arrayen:

```
exceptionValue: ReferenceError: Can't find variable: melbaToast
sourceURL: app:/test.html
line: 5
function: throwError
sourceURL: app:/test.html
line: 10
function: onclick
```

Hantera körningshändelser med JavaScript

Adobe AIR 1.0 och senare

Körningsklasserna kan lägga till händelsehanterare med metoden `addEventListener()`. Om du vill lägga till en hanterarfunktion för en händelse anropar du metoden `addEventListener()` för det objekt som skickar händelsen och som innehåller händelsetypen och hanteringsfunktionen. Om du till exempel vill avlyssna den `closing`-händelse som skickas när en användare klickar på fönstrets stängningsknapp i namnlisten, använder du följande programsats:

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

Skapa en händelsehanterarfunktion

Adobe AIR 1.0 och senare

Följande kod skapar en enkel HTML-fil som visar information om huvudfönstrets placering. En hanterarfunktion med namnet `moveHandler()` avlyssnar en move-händelse (definieras av klassen NativeWindowBoundsEvent) i huvudfönstret.

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

Om en användare flyttar fönstret visar textarea-elementen fönstrets uppdaterade X- och Y-positioner:

Observera att händelseobjektet överförs som ett argument till metoden `moveHandler()`. Med hjälp av händelseparametern kan hanterarfunktionen undersöka händelseobjektet. I det här exemplet använder du händelseobjektets `type`-egenskap för att rapportera att händelsen är en `move`-händelse.

Ta bort händelseavlyssnare

Adobe AIR 1.0 och senare

Du kan använda metoden `removeEventListener()` när du vill ta bort en händelseavlyssnare som inte längre behövs. Det kan vara en bra idé att ta bort avlyssnare som inte längre används. Parametrar som krävs innefattar parametrarna `eventName` och `listener`, som är desamma som de parametrar som krävs för metoden `addEventListener()`.

Ta bort händelseavlyssnare på HTML-sidor som innehåller navigering

Adobe AIR 1.0 och senare

När navigering görs i HTML-innehåll, eller när HTML-innehåll ignoreras på grund av att fönstret som innehållet finns i stängs, tas inte de händelseavlyssnare som refererar till objekten på den ej inlästa sidan bort automatiskt. Om ett objekt skickar en händelse till en hanterare som redan har tagits bort visas ett felmeddelande av följande typ: "Programmet försökte referera till ett JavaScript-objekt på en HTML-sida som inte finns inläst."

För att undvika det här felet tar du bort JavaScript-händelseavlyssnare på en HTML-sida innan sidan tas bort. Vid sidnavigering (i ett HTMLLoader-objekt) tar du bort händelseavlyssnaren under `unload`-händelsen för `window`-objektet.

Följande JavaScript-kod tar till exempel bort en händelseavlyssnare för en `uncaughtScriptException`-händelse:

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Om du vill förhindra att det här felet uppstår när fönster som innehåller HTML-innehåll stängs, så anropar du en cleanup-funktion som svar på `closing`-händelsen för NativeWindow-objektet (`window.nativeWindow`). Följande JavaScript-kod tar till exempel bort en händelseavlyssnare för en `uncaughtScriptException`-händelse:

```
 window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Du kan också förhindra att det här felet inträffar genom att ta bort en händelseavlyssnare så snart den körs (om händelsen bara behöver hanteras en gång). Följande JavaScript-kod skapar till exempel ett html-fönster genom att anropa metoden `createRootWindow()` för klassen HTMLLoader och lägger till en händelseavlyssnare för `complete`-händelsen. När `complete`-händelsehanteraren anropas tar den bort sin egen händelseavlyssnare med hjälp av `removeEventListener()`-funktionen:

```
 var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

Om du tar bort onödiga händelseavlyssnare kan systemets skräpinsamlare frigöra det minne som var kopplat till dessa avlyssnare.

Söka efter befintliga händelseavlyssnare

Adobe AIR 1.0 och senare

Med hjälp av metoden `hasEventListener()` kan du söka efter befintliga händelseavlyssnare i ett objekt.

Kapitel 62: Visa HTML-innehåll i mobilprogram

Adobe AIR 2.5 och senare

Klassen StageWebView visar HTML-innehåll med hjälp av systemets webbläsarkontroll på mobilenheter och med hjälp av standardkontrollen HTMLLoader i Adobe® AIR® på stationära datorer. Kontrollera egenskapen `StageWebView.isSupported` för att avgöra om klassen stöds på den aktuella enheten. Det är inte säkert att alla enheter stöds i mobilprofilen.

Oavsett profil har klassen StageWebView bara stöd för begränsad interaktion mellan HTML-innehållet och resten av programmet. Du kan styra navigeringen, men varken korsskriptning eller direktutbyte av data tillåts. Du kan läsa in innehållet från en lokal URL eller en fjärr-URL eller skicka en HTML-sträng.

StageWebView-objekt

Ett StageWebView-objekt är inget visningsobjekt och kan inte läggas till i visningslistan. Det styr i stället en visningsruta som är kopplad direkt till scenen. StageWebView-innehåll ritas ovanpå allt annat innehåll i visningslistan. Det går inte att styra ritningsordningen för flera StageWebView-objekt.

Om du vill visa ett StageWebView-objekt tilldelar du den scen, på vilken objektet ska visas, till `stage`-egenskapen för StageWebView. Ange storleken på visningen med hjälp av `viewPort`-egenskapen.

Ange x- och y-koordinaterna för `viewPort`-egenskapen mellan -8 192 och 8 191. Det största tillåtna värdet på scenens bredd och höjd är 8 191. Om storleken överskrider maxvärdena genereras ett undantagsfel.

I följande exempel skapas ett StageWebView-objekt, egenskaperna `stage` och `viewPort` anges och en HTML-sträng visas:

```
var webView:StageWebView = new StageWebView();
webView.viewPort = new Rectangle( 0, 0, this.stage.stageWidth, this .stage.stageHeight);
webView.stage = this.stage;
var htmlString:String = "<!DOCTYPE HTML>" +
                        "<html><body>" +
                        "<p>King Philip could order five good steaks.</p>" +
                        "</body></html>";
webView.loadString( htmlString );
```

Om du vill dölja ett StageWebView-objekt anger du dess `stage`-egenskap som `null`. Om du vill kasta bort objektet helt anropar du metoden `dispose()`. Du måste inte anropa `dispose()`, men det bidrar till att skräpinsamlaren kan återta det minnesutrymme som användes av objektet fortare.

Innehåll

Du kan läsa in innehåll till ett StageWebView-objekt med två metoder: `loadURL()` och `loadString()`.

Metoden `loadURL()` läser in en resurs vid den angivna URL:en. Du kan använda alla URI-scheman som stöds av systemets webbläsarkontroll, bland annat data:, file:, http:, https: och javascript:. Schemana app: och app-storage: stöds inte. AIR validerar inte URL-strängen.

Metoden `loadString()` läser in en litteral sträng med HTML-innehåll. Platsen för en sida som läses in med den här metoden uttrycks som:

- På skrivbordet: about:blank
- I iOS: *htmlString*
- I Android: URI-dataformat för den kodade *htmlString*

URI-schemat avgör vilka regler som används för att läsa in inbäddat innehåll eller inbäddade data.

URI-schema	Inläsning av lokal resurs	Inläsning av fjärresurs	Lokal XMLHttpRequest	Fjärr-XMLHttpRequest
data:	Nej	Ja	Nej	Nej
file:	Ja	Ja	Ja	Ja
http:, https:	Nej	Ja	Nej	Samma domän
about: (loadString()-metod)	Nej	Ja	Nej	Nej

Obs! *Om scenens `displayState`-egenskap är inställd på `FULL_SCREEN` kan du, på skrivbordet, inte skriva i ett textfält som visas i StageWebView. I iOS och Android kan du däremot skriva i ett textfält på StageWebView, även om scenens `displayState` är `FULL_SCREEN`.*

I följande exempel används ett StageWebView-objekt för att visa Adobes webbplats:

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );
        }
    }
}
```

På Android-enheter måste du ange INTERNET-behörighet för att programmet ska lyckas läsa in fjärresurser.

I Android 3.0+ måste maskinvaruaccelerationen aktiveras med ett program i Android-elementet manifestAdditions i AIR:s programbeskrivning för att plugin-innehåll i ett StageWebView-objekt ska visas. Se Aktivera Flash Player och andra plugin-program i ett StageWebView-objekt.

JavaScript-URI

Du kan använda en JavaScript-URI för att anropa en funktion som definierats på den HTML-sida som läses in av ett StageWebView-objekt. Den funktion som anropas med JavaScript-URI:n körs i den inlästa webbsidans sammanhang. I följande exempel används ett StageWebView-objekt för att anropa en JavaScript-funktion:

```
package {
    import flash.display.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    public class WebView extends Sprite
    {
        public var webView:StageWebView = new StageWebView();
        public function WebView()
        {
            var htmlString:String = "<!DOCTYPE HTML>" +
            "<html><script type=text/javascript>" +
            "function callURI(){" +
            "alert(\"You clicked me!!\");"+
            "}</script><body>" +
            "<p><a href=javascript:callURI()>Click Me</a></p>" +
            "</body></html>";
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadString( htmlString );
        }
    }
}
```

Fler hjälpavsnitt

Sean Voisen: Making the Most of StageWebView (Utnyttja StageWebView optimalt)

Navigeringshändelser

När en användare klickar på en länk i HTML-koden skickar StageWebView-objektet en `locationChanging`-händelse. Du kan anropa metoden `preventDefault()` för händelseobjektet om du vill stoppa navigeringen. Annars navigerar StageWebView-objektet till den nya sidan och skickar en `locationChange`-händelse. När sidan har lästs in skickar StageWebView-objektet en `complete`-händelse.

En `locationChanging`-händelse skickas för varje HTML-omdirigering. Händelserna `locationChange` och `complete` skickas vid lämplig tidpunkt.

I iOS skickas en `locationChanging`-händelse före en `locationChange`-händelse, utom för den första `loadURL()`- eller `loadString()`-metoden. En `locationChange`-händelse skickas även för navigeringsändringar via iframes och bildrutor.

Följande exempel visar hur du kan förhindra en platsförändring och öppna den nya sidan i systemets webbläsare i stället.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.events.LocationChangeEvent;
    import flash.geom.Rectangle;
    import flash.net.navigateToURL;
    import flash.net.URLRequest;

    public class StageWebViewNavEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();

        public function StageWebViewNavEvents() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
        webView.addEventListener( LocationChangeEvent.LOCATION_CHANGING, onLocationChanging );
            webView.loadURL( "http://www.adobe.com" );
        }
        private function onLocationChanging( event:LocationChangeEvent ):void
        {
            event.preventDefault();
            navigateToURL( new URLRequest( event.location ) );
        }
    }
}
```

Historik

När användaren klickar på länkar i det innehåll som visas i StageWebView-objektet sparas framåt- och bakåthistoriken. Följande exempel visar hur du navigerar genom de båda historikstackarna. I exemplet används valknapparna för Bakåt och Framåt.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample()
        {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );

            stage.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            if( event.keyCode == Keyboard.BACK && webView.isHistoryBackEnabled )
            {
                trace("back");
                webView.historyBack();
                event.preventDefault();
            }
            if( event.keyCode == Keyboard.SEARCH && webView.isHistoryForwardEnabled )
            {
                trace("forward");
                webView.historyForward();
            }
        }
    }
}
```

Fokus

Även om StageWebView-klassen inte är något visningsobjekt innehåller den medlemmar som du kan använda för att hantera fokusövergångar till och från kontrollen.

När StageWebView-objektet får fokus skickar det en `focusIn`-händelse. Du använder den här händelsen för att hantera fokuselement i programmet, om det behövs.

När StageWebView-objektet tappar fokus skickar det en `focusOut`-händelse. En StageWebView-instans kan släppa fokus när en användare ”tabbar förbi” den första eller sista kontrollen på webbsidan med en styrkula eller med riktningspilar. Med egenskapen `direction` för händelseobjektet kan du få veta om fokusflödet går uppåt förbi sidans överkant eller nedåt förbi dess nederkant. Använd den här informationen för att ge rätt visningsobjekt fokus ovanför eller under StageWebView.

I iOS kan du inte ange fokus i programmet. StageWebView skickar `focusIn`- och `focusOut`-händelser med direction-egenskapen för `FocusEvent` inställd på `none`. Om användaren vidrör något inom StageWebView skickas `focusIn`-händelsen. Om användaren vidrör något utanför StageWebView skickas `focusOut`-händelsen.

Följande exempel visar hur fokus skiftar från StageWebView-objektet till Flash-visningsobjekten:

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.FocusEvent;
    import flash.display.FocusDirection;
    import flash.events.LocationChangeEvent;

    public class StageWebViewFocusEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();
        var topControl:TextField = new TextField();
        var bottomControl:TextField = new TextField();

        public function StageWebViewFocusEvents()
        {
            trace("Starting");
            topControl.type = TextFieldType.INPUT;
            addChild( topControl );
            topControl.height = 60;
            topControl.width = stage.stageWidth;
            topControl.background = true;
            topControl.text = "One control on top.";
            topControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            topControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );

            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 60, stage.stageWidth, stage.stageHeight
- 120 );
            webView.addEventListener( FocusEvent.FOCUS_IN, webFocusIn );
            webView.addEventListener(FocusEvent.FOCUS_OUT, webFocusOut );
            webView.addEventListener(LocationChangeEvent.LOCATION_CHANGING,
                                     function( event:LocationChangeEvent ):void
                                     {
                                        event.preventDefault();
                                     } );
            webView.loadString("<form action='#'><input/><input/><input/></form>");
            webView.assignFocus();

            bottomControl.type = TextFieldType.INPUT;
            addChild( bottomControl );
            bottomControl.y = stage.stageHeight - 60;
            bottomControl.height = 60;
            bottomControl.width = stage.stageWidth;
            bottomControl.background = true;
            bottomControl.text = "One control on the bottom.";
            bottomControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            bottomControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );}

        private function webFocusIn( event:FocusEvent ):void
        {
            trace("Web focus in");
```

```
        }

        private function webFocusOut( event:FocusEvent ):void
        {
            trace("Web focus out: " + event.direction);
            if( event.direction == FocusDirection.TOP )
            {
                stage.focus = topControl;
            }
            else
            {
                stage.focus = bottomControl;
            }
        }

        private function flashFocusIn( event:FocusEvent ):void
        {
            trace("Flash focus in");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xff5566;
        }

        private function flashFocusOut( event:FocusEvent ):void
        {
            trace("Flash focus out");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xffffff;
        }

    }
}
```

Ta bitmappsbilder

Ett StageWebView-objekt återges ovanpå allt annat innehåll i visningslistan. Du kan inte lägga till innehåll ovanpå ett StageWebView-objekt. Du kan till exempel inte utöka en listruta så att den sträcker sig över StageWebView-innehållet. Det här problemet kan du åtgärda genom att ta en ögonblicksbild av StageWebView. Sedan döljer du StageWebView och lägger till ögonblicksbilden i stället.

Följande exempel visar hur du tar en ögonblicksbild av ett StageWebView-objekt med metoden `drawViewPortToBitmapData`. StageWebView-objektet döljs genom att scenen ställs in på null. När webbsidan har lästs in helt anropas en funktion som tar en ögonblicksbild av bitmappen och visar den. När koden körs visas två etiketter, Google och Facebook. När användaren klickar på en etikett hämtas motsvarande webbsida och visas som en ögonblicksbild på scenen.

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    import flash.net.*;
    import flash.text.TextField;
    public class stagewebview extends Sprite
    {
        public var webView:StageWebView=new StageWebView();
        public var textGoogle:TextField=new TextField();
        public var textFacebook:TextField=new TextField();
        public function stagewebview()
        {
            textGoogle.htmlText="<b>Google</b>";
            textGoogle.x=300;
            textGoogle.y=-80;
            addChild(textGoogle);
            textFacebook.htmlText="<b>Facebook</b>";
            textFacebook.x=0;
            textFacebook.y=-80;
            addChild(textFacebook);
            textGoogle.addEventListener(MouseEvent.CLICK,goGoogle);
            textFacebook.addEventListener(MouseEvent.CLICK,goFaceBook);
            webView.stage = this.stage;
            webView.viewPort = new Rectangle(0, 0, stage.stageWidth, stage.stageHeight);
        }
    public function goGoogle(e:Event):void
        {
            webView.loadURL("http://www.google.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }

    public function goFaceBook(e:Event):void
        {
            webView.loadURL("http://www.facebook.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }
    public function handleLoad(e:Event):void
        {
            var bitmapData:BitmapData = new BitmapData(webView.viewPort.width,
webView.viewPort.height);
            webView.drawViewPortToBitmapData(bitmapData);
            var webViewBitmap:Bitmap=new Bitmap(bitmapData);
            addChild(webViewBitmap);
        }
    }
}
```

Kapitel 63: Använda arbetare för samtidighet

Flash Player 11.4 och senare, Adobe AIR 13.4 och senare för datorplattformar

Med ActionScript-arbetare är det möjligt att köra kod samtidigt, eller med andra ord, köra kod i bakgrunden utan att störa kodkörningen i huvudprogrammet.

API:er för ActionScript-samtidighet på datorplattformar är endast tillgängliga i Flash Player 11.4 och senare och i AIR 3.4 och senare. Samtidighet stöds inte i AIR för mobilplattformar.

Arbeta med arbetare och samtidighet

Flash Player 11.4 och senare, Adobe AIR 13.4 och senare för datorplattformar

När ett program inte använder arbetare körs programmets kod linjärt i steg som kallas *körningstråd*. I tråden utförs den kod som utvecklaren har skrivit. Dessutom körs kod som är en del av miljön, vanligtvis kod som uppdaterar skärmen när egenskaper för visningsobjekten ändras. Trots att kod skrivs i segment som metoder och klasser kommer koden vid körningen att exekveras en rad i taget som om den var skriven i en enda lång serie med steg. Här följer ett hypotetiskt exempel på steg som utförs i ett program:

1. Öppna bildruta: I miljön anropas en `enterFrame`-händelsehanterare och kör koden i den, en rad i taget
2. Mushändelse: Användaren rör musen och i miljön anropas mushändelsehanteraren när olika rollover- och rollout-händelser inträffar
3. Läser in slutförd händelse: En begäran om att läsa in en xml-fil från en url returneras med inlästa fildata. Händelsehanteraren anropas och stegen i den körs, xml-innehållet läses in och en objektuppsättning skapas från xml-data.
4. Mushändelse: Musen har på nytt flyttats, så relevant mushändelsehanterare anropas
5. Återgivning: Inga fler händelser förväntas, så skärmen uppdateras utifrån ändringar gjorda i visningsobjekt
6. Öppna bildruta: Kodcykeln återupptas

Så som det beskrivs i exemplet körs de hypotetiska stegen 1 till 5 i en sekvens inom ett tidsblock som kallas en bildruta. Eftersom de körs i en sekvens i en tråd går det inte att avbryta ett steg i processen för att köra ett annat. Med en frekvens på 30 bildrutor per sekund finns det endast tid för att utföra alla dessa operationer under en trettiondels sekund. I de flesta fall är det tillräckligt för att hinna köra koden och miljön väntar under den återstående tiden. Anta emellertid att de xml-data som läses in i steg 3 är mycket stora och djupt inkapslade i xml-strukturen. När koden körs i en slinga i xml-koden och skapar objekt kan det ta längre tid än en trettiondels sekund att utföra denna uppgift. I detta fall kommer de återstående stegen (svar på mushändelser och omritning av skärmen) inte att inträffa så snabbt som de borde. Detta leder till att skärmen låser sig och att dataströmmen blir ojämn så att den inte ritas om tillränkligt snabbt för användaren som rör musen.

Om all kod körs i samma tråd finns det bara ett sätt att undvika ojämna dataströmmar och frysningar. Motdraget är att inte skapa åtgärder som tar lång tid, såsom slingor över stora datamängder. Med ActionScript-arbetare erbjuds du en annan lösning. Om du använder arbetare kan du göra omfattande kodkörningar i en separat arbetare. Varje arbetare körs i en separat tråd så att den omfattande kodkörningen utförs i en egen tråd i bakgrunden. Detta frigör arbetarens huvudprogram så att skärmen kan ritas om för varje bildruta utan att blockera annat arbete.

Möjligheten att utföra flera kodoperationer samtidigt på detta sätt kallas för *samtidighet* (concurrency). När bakgrundsarbetaren är klar med sitt arbete eller medan det pågår, kan du skicka meddelanden och data för huvudarbetaren. Detta medför att du kan skriva kod som utför komplexa och tidskrävande operationer, men slippa nackdelar så som att skärmen fryser.

Arbetare är värdefulla eftersom de minskar riskerna för bildrutebortfall på grund av att huvudåtergivningstråden blockerats av annan kod. Emellertid kräver arbetare mer systemminne och processorkapacitet vilket kan påverka programmets generella prestanda negativt. Eftersom varje arbetare använder egna instanser av den virtuella miljön kan även pålägget för en enkel arbetare vara stort. När du använder arbetare ska du testa koden på alla målplattformar för att försäkra dig om att inte belastningen på systemet blir för stor. Adobe rekommenderar att du inte använder mer än en eller två bakgrundsarbetare i vanliga situationer.

Skapa och hantera arbetare

Flash Player 11.4 och senare, Adobe AIR 13.4 och senare för datorplattformar

Det första steget när du vill använda en arbetare för samtidighet är att skapa en bakgrundsarbetare. Du använder två typer av objekt när du skapar en arbetare. Den första som du skapar är Worker-instansen. Den andra är ett WorkerDomain-objekt, som skapar arbetaren och administrerar hur Worker-objekten körs i ett program.

När miljön läses in skapas automatiskt WorkerDomain-objektet. I miljön skapas dessutom automatiskt en arbetare för huvud-swf-filen för programmet. Den första arbetaren kallas den *primitiva arbetaren*.

Eftersom det bara finns ett WorkerDomain-objekt för ett program får du åtkomst till WorkerDomain-instansen via den statiska `WorkerDomain.current`-egenskapen.

Du har när som helst åtkomst till den aktuella Worker-instansen (arbetaren i vilken den aktuella koden körs) via den statiska `Worker.current`-egenskapen.

Skapa ett Worker-objekt från en swf-fil

Precis som huvud-swf-filen körs i den primitiva arbetaren, kör bakgrundsarbetaren kod från en enskild swf-fil. Om du vill använda en bakgrundsarbetare måste du skapa och kompilera koden som en swf-fil. Om du vill skapa bakgrundsarbetaren måste den överordnade arbetaren ha åtkomst till swf-filens byte som ett ByteArray-objekt. Du skickar denna ByteArray till WorkerDomain-objektets `createWorker()`-metod för att de facto skapa arbetaren.

Det finns tre sätt att erhålla bakgrundsarbetarens swf-fil som ett ByteArray-objekt:

Bädda in arbetarens swf-fil

Använd metataggen [Embed] för att bädda in arbetarens swf-fil i huvud-swf-filen som en ByteArray:

```
[Embed(source="../swfs/BgWorker.swf", mimeType="application/octet-stream")]
private static var BgWorker_ByteClass:Class;
private function createWorker():void
{
    var workerBytes:ByteArray = new BgWorker_ByteClass();
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

Arbetarens swf-fil kompileras in i huvud-swf-filen som en ByteArray-underklass med namnet BgWorker_ByteClass. Genom att skapa en instans av klassen får du en ByteArray ifylld med byte från arbetarens swf-fil.

Läsa in en externa swf-fil för arbetaren

Använd ett URLLoader-objekt för att läsa in en extern swf-fil för arbetaren. Swf-filen måste komma från samma säkerhetsdomän, till exempel som en swf-fil som läses in från samma internetdomän som huvud-swf-filen, eller ingå i ett AIR-programpaket.

```
var workerLoader:URLLoader = new URLLoader();
workerLoader.dataFormat = URLLoaderDataFormat.BINARY;
workerLoader.addEventListener(Event.COMPLETE, loadComplete);
workerLoader.load(new URLRequest("BgWorker.swf"));

private function loadComplete(event:Event):void
{
    // create the background worker
    var workerBytes:ByteArray = event.target.data as ByteArray;
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

När inläsningen av swf-filen med URLLoader är klar är swf-filens byte tillgängliga i URLLoader-objektets `data`-egenskap (`event.target.data` i detta exempel).

Använda huvud-swf-filen som swf-fil för arbetaren

Du kan använda en enskild swf-fil som både huvud-swf-fil och som swf-fil för arbetaren. Använd huvudvisningsklassens `loaderInfo.bytes`-egenskap för att få åtkomst till swf-filens byte.

```
// The primordial worker's main class constructor
public function PrimordialWorkerClass()
{
    init();
}

private function init():void
{
    var swfBytes:ByteArray = this.loaderInfo.bytes;

    // Check to see if this is the primordial worker or the background worker
    if (Worker.current.isPrimordial)
    {
        // create a background worker
        var bgWorker:Worker = WorkerDomain.current.createWorker(swfBytes);

        // ... set up worker communication and start the worker
    }
    else // entry point for the background worker
    {
        // set up communication between workers using getSharedProperty()
        // ... (not shown)

        // start the background work
    }
}
```

Om du använder denna teknik ska du använda en `if`-sats för att dela upp koden för swf-filen i huvudklassens konstruktor eller i den metod som den anropar. För att kunna bestämma om koden körs i huvudarbetaren eller i bakgrundsarbetaren ska du kontrollera det aktuella Worker-objektets `isPrimordial`-egenskap, se exemplet nedan.

Starta körning av en arbetare

När du har skapat en arbetare startar du körningen av den genom att anropa Worker-objektets `start()`-metod. Åtgärden `start()` startar inte omedelbart. Om du vill veta när arbetaren körs ska du registrera en avlyssnare för Worker-objektets `workerState`-händelse. Denna händelse skickas när Worker-objektet ändrar lägen under livscykeln, till exempel när kodkörningen inleds. I händelsehanteraren `workerState` kontroller du att Worker-objektets `state`-egenskap är `WorkerState.RUNNING`. Du gör det när arbetaren körs och när huvudklassens konstruktor har körts. I följande kod visas ett exempel på registrering för `workerState`-händelsen och anropet av `start()`-metoden:

```
// listen for worker state changes to know when the worker is running
bgWorker.addEventListener(Event.WORKER_STATE, workerStateHandler);
// set up communication between workers using
// setSharedProperty(), createMessageChannel(), etc.
// ... (not shown)
bgWorker.start();
private function workerStateHandler(event:Event):void
{
    if (bgWorker.state == WorkerState.RUNNING)
    {
        // The worker is running.
        // Send it a message or wait for a response.
    }
}
```

Administrera körning av arbetare

Med WorkerDomain-klassens `listWorkers()`-metod kan du när som helst komma åt den uppsättning arbetare som körs i programmet. Den här metoden returnerar den uppsättning arbetare vars `state`-egenskap är `WorkerState.RUNNING`, inklusive den primitiva arbetaren. Den är inte inkluderad om arbetaren inte har startats eller om körningen redan har stoppats.

Om du inte längre behöver en arbetare ska du anropa Worker-objektets `terminate()`-metod för att stänga arbetaren och frisläppa dess minne och andra systemresurser.

Kommunicera mellan arbetare

Flash Player 11.4 och senare, Adobe AIR 13.4 och senare för datorplattformar

Trots att arbetare kör respektive kod i separata huvudprogram så vore det ingen fördel om de var fullständigt isolerade från varandra. Kommunikationen mellan arbetare innebär i sista hand att data skickas mellan dem. Det finns tre metoder för att hämta data från en arbetare och skicka dem till en annan.

När du ska avgöra vilken av dessa datadelningstekniker som är lämpligast för att skicka vissa data ska du beakta de två viktigaste skillnaderna. En skillnad mellan dem är om det är en händelse som ska meddela mottagaren om att nya data är tillgängliga eller om det är den mottagande arbetaren som ska kontrollera om det finns uppdateringar. En annan skillnad mellan dessa datadelningstekniker har att göra med hur data de facto skickas. I vissa fall erhåller den mottagande arbetaren en kopia av delade data, vilket innebär att fler objekt skapas och att mer minnesutrymme och processorkapacitet krävs. I vissa fall erhåller arbetarna objekt som refererar till samma underliggande systemminne, vilket innebär att färre objekt skapas och i ett generellt mindre minnesutnyttjande. Dessa skillnader beskrivs här:

Kommunikatio nsteknik	Skickar händelse när data erhålls	Delar minne mellan arbetare
Delade egenskaper mellan arbetare	Nej	Nej, objekt kopieras inte referenser
MessageChannel	Ja	Nej, objekt kopieras inte referenser
Delbar ByteArray	Nej	Ja, minnet delas

Skicka data med en delad egenskap

Det enklaste sättet att dela data mellan arbetare är att använda en delad egenskap. Varje arbetare underhåller en intern ordlista med delade egenskapsvärden. Dessa egenskaper är lagrade med strängnyckelnamn för att skilja dem från egenskaper. Om du vill lagra ett objekt i en arbetare som en delad egenskap anropar du Worker-objektets `setSharedProperty()`-metod med två argument, nyckelnamnet och värdet som ska lagras:

```
// code running in the parent worker
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

När den delade egenskapen är inställd kan värdet läsas genom ett anrop av Worker-objektets `getSharedProperty()`-metod, skickat i nyckelnamnet:

```
// code running in the background worker
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

Det finns inga restriktioner för vilken arbetare som läser eller ställer in egenskapsvärdet. Koden i bakgrundsarbetaren kan till exempel anropa dess `setSharedProperty()`-metod för att dela ett värde. Kod som körs i den överordnade arbetaren kan sedan använda `getSharedProperty()` för att hämta data.

Värdet som skickas till metoden `setSharedProperty()` kan näst intill vara alla typer av objekt. När du anropar metoden `getSharedProperty()` är det objekt som returneras en kopia av objektet som skickades till `setSharedProperty()` och inte en referens till samma objekt. Detta gäller med några få undantag. I "Delade referenser och kopierade värden" på sidan 1045 förklaras mer ingående hur data delas.

Den största fördelen med att använda en delad egenskap för att skicka data mellan arbetare är att den är tillgänglig även innan arbetaren körs. Du kan anropa Worker-objektets `setSharedProperty()`-metod i bakgrunden för att ställa in en delad egenskap även innan arbetaren körs. När den överordnade arbetaren anropar arbetarens `start()`-metod, kommer den underordnade arbetarens huvudklasskonstruktor att anropas. Eventuella delade egenskaper som ställdes in innan `start()` anropades är tillgängliga för kod som kan läsas i den underordnade arbetaren.

Skicka data med MessageChannel

En meddelandekanel är en envägslänk för att skicka data mellan två arbetare. Det finns en viktig fördel med att använda ett MessageChannel-objekt för att skicka data mellan arbetare. När du skickar ett meddelande (ett objekt) med en meddelandekanal kommer MessageChannel-objektet att skicka en `channelMessage`-händelse. Kod i den mottagande arbetaren kan avlyssna den händelsen för att veta när data är tillgängligt. Detta innebär att den mottagande arbetaren inte kontinuerligt behöver kontrollera om det finns datauppdateringar.

En meddelandekanal är associerad med endast två arbetare, en avsändare och en mottagare. Om du vill skapa ett MessageChannel-objekt, anropar du Worker-objektets `createMessageChannel()`-metod, genom att skicka den mottagande arbetaren som ett argument:

```
// In the sending worker swf
var sendChannel:MessageChannel;
sendChannel = Worker.current.createMessageChannel(receivingWorker);
```

Båda arbetarna måste ha åtkomst till MessageChannel-objektet. Det enklaste sättet att göra detta att skicka MessageChannel-objektet med metoden `setSharedProperty()`:

```
receivingWorker.setSharedProperty("incomingChannel", sendChannel);
```

Registrera i den mottagande arbetaren en avlyssnare för MessageChannel-objektets `channelMessage`-händelse. Denna händelse skickas när den avsändande arbetaren skickar data genom meddelandekanalen.

```
// In the receiving worker swf
var incomingChannel:MessageChannel;
incomingChannel = Worker.current.getSharedProperty("incomingChannel");
incomingChannel.addEventListener(Event.CHANNEL_MESSAGE, handleIncomingMessage);
```

För att faktiskt kunna skicka data ska du i den avsändande arbetaren anropa MessageChannel-objektets `send()` metod:

```
// In the sending worker swf
sendChannel.send("This is a message");
```

I den mottagande arbetaren anropar MessageChannel händelsehanteraren för `channelMessage`. Den mottagande arbetaren kan sedan hämta data genom att anropa MessageChannel-objektets `receive()`-metod.

```
private function handleIncomingMessage(event:Event):void
{
    var message:String = incomingChannel.receive() as String;
}
```

Objektet som returneras av metoden receive har samma datatyp som objektet som skickas till metoden `send()`. Det mottagna objektet är en kopia av objektet som skickades av avsändaren och inte en referens till objektet i den avsändande arbetaren, såvida det inte tillhör någon av de få datatyperna som beskrivs i "Delade referenser och kopierade värden" på sidan 1045.

Dela data med en delbar ByteArray

När ett objekt skickas mellan två arbetare erhåller den mottagande arbetaren ett nytt objekt som är en kopia av originalet. De båda objekten är lagrade på olika platser i systemets minne. Detta innebär att varje kopia av objektet som erhålls ökar den totala minneskapaciteten som används under körningen. Dessutom så innebär det att varje ändring som du gör i ett objekt i en arbetare inte påverkar kopian i den andra arbetaren. Mer information om hur data kopieras finns i "Delade referenser och kopierade värden" på sidan 1045.

Standard är att samma beteende används för ett ByteArray-objekt. Om du skickar en ByteArray-instans till en `setSharedProperty()`-metod för ett Worker-objekt eller till en `send()`-metod för ett MessageChannel-objekt, skapas under körningen en ny ByteArray i datorns minne och den mottagande arbetaren erhåller en ByteArray-instans som är en referens till den nya ByteArray-instansen. Du kan emellertid ändra detta beteende för ett ByteArray-objekt genom att ange att `shareable`-egenskapen ska vara `true`.

När ett delbart ByteArray-objekt skickas från en arbetare till en annan, är ByteArray-instansen i den mottagande arbetaren en referens till samma underliggande systemminne som används för ByteArray-instansen i den avsändande arbetaren. När koden för en arbetare ändrar innehållet för bytearrayen kommer dessa ändringar omedelbart att vara tillgängliga i andra arbetare som har åtkomst till den delade bytearrayen.

Eftersom koden för arbetarna körs samtidigt är det möjligt för två arbetare att få åtkomst till samma byte i en bytearray samtidigt. Detta kan leda till dataförluster och andra fel. Det finns flera API:er som du kan använda för att hantera åtkomsten till delade resurser och undvika dessa problem.

I klassen ByteArray finns metoder för att validera och ändra bytearrayens innehåll med en enskild operation:

- atomicCompareAndSwapIntAt() metod
- atomicCompareAndSwapLength() metod

Dessutom innehåller flash.concurrent-paketet klasser som erbjuder åtkomstkontroll för att arbeta med delade resurser:

- Mutex klass
- Condition klass

Delade referenser och kopierade värden

I normalfallet när du anropar `Worker.setSharedProperty()` eller `MessageChannel.send()`, har objektet som skickas till den mottagande arbetaren ett serialiserat AMF-format. Detta får vissa konsekvenser:

- Objektet som skapats i den mottagande arbetaren när dess `getSharedProperty()`-metod anropas kommer att avserialiseras från AMF-byte. Det är en kopia av originalobjektet och inte en referens till objektet. Ändringar som görs i objektet i någon av arbetarna ändras inte i kopian hos den andra arbetaren.
- Objekt som inte kan serialiseras i AMF-format såsom visningsobjekt, kan inte skickas till en arbetare med `Worker.setSharedProperty()` eller `MessageChannel.send()`.
- För att en anpassad klass ska kunna avserialiseras måste klassdefinitionen registreras med `flash.net.registerClassAlias()`-funktionen eller med `[RemoteClass]`-metadata. Samma alias måste användas för båda arbetarnas versioner av klassen.

Det finns fem specialobjekt som verkligen delas, och inte kopieras, mellan arbetare:

- Worker-objekten
- MessageChannel-objekten
- delbar bytearray (ett ByteArray-objekt vars `shareable`-egenskap är `true`)
- Mutex-objekt
- Condition-objekt

När du skickar en instans av dessa objekt med `Worker.setSharedProperty()`- eller `MessageChannel.send()`-metoden får varje arbetare en referens till samma underliggande objekt. Ändringar gjorda i en instans i en arbetare är omedelbart tillgängliga i en annan arbetare. Om du dessutom skickar samma instans av dessa objekt till en arbetare mer än en gång, kommer ingen ny kopia av objektet att skapas i den mottagande arbetarens miljö. I stället kommer samma referens att återanvändas.

Fler datadelningstekniker

Förutom arbetarspecifika metoder för att skicka data, kan arbetare även utbyta data genom att använda någon av de befintliga API:erna som har stöd för att dela data mellan två swf-program, som exempelvis följande:

- lokalt delade objekt
- skriva data i en fil för en arbetare och läsa från filen i den andra arbetaren
- lagra data i och läsa data från en SQLite-databas

När du delar en resurs mellan två eller flera arbetare bör du vanligtvis undvika att låta flera arbetare få åtkomst till resursen samtidigt. Om du till exempel låter flera arbetare få åtkomst till en fil i ett lokalt filsystem kan detta medföra dataförluster eller fel, och det kanske inte heller stöds i operativsystemet.

Om du vill skydda dig mot samtidiga åtkomstproblem ska du använda klasserna Mutex och Condition i flash.concurrent-paketet för att få kontrollen över detta när du arbetar med delade resutser.

Till skillnad från andra datadelningsmetoder är SQLite-databasmotorn framtagen för samtidig åtkomst och den har ett eget internt stöd för transaktioner. Flera arbetare kan samtidigt använda en SQLite-databas utan risk för att förstöra några data. Eftersom arbetarna använder olika SQLConnection-instanser får varje arbetare åtkomst till databasen via en separat transaktion. Samtidiga datahanteringsoperationer påverkar inte dataintegriteten.

Se även

”Arbeta med lokala SQL-databaser i AIR” på sidan 716
flash.concurrent paket

Kapitel 64: Säkerhet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Säkerhet är en huvudfråga för Adobe, användare, webbplatsägare och innehållsutvecklare. Av det skälet innehåller Adobe® Flash® Player och Adobe® AIR™ en uppsättning säkerhetsregler och -kontroller för att skydda användaren, webbsideägaren och innehållsutvecklaren. Den här informationen omfattar säkerhetsmodellen för SWF-filer som publiceras med ActionScript 3.0 och körs i Flash Player 9.0.124.0 eller senare, och SWF-, HTML- och JavaScript-filer som körs i AIR 1.0 eller senare, om inget annat anges.

Det här avsnittet är avsett att ge en översikt över säkerheten och innehåller inga utförliga beskrivningar av implementeringsdetaljer, användarscenarier eller användning av vissa API:er. Mer information om säkerhetsfrågor i Flash Player finns i avsnittet om säkerhet på Player Developer Center på www.adobe.com/go/devnet_security_se.

Säkerhet med Flash Platform – översikt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En stor del av säkerhetsmodellen som används av körtiderna i Flash Player och AIR bygger på ursprungsdomänen för inlästa SWF-filer, HTML, media och andra resurser. Körbar kod i en fil från en specifik Internetdomän, till exempel www.example.com, kan alltid få åtkomst till alla data från den domänen. Dessa resurser placeras i samma säkerhetsgruppering, även kallad *säkerhetssandlåda*. (Mer information finns i ”Säkerhetssandlådor” på sidan 1049.)

Exempelvis kan en ActionScript-kod i en SWF-fil läsa in SWF-filer, bitmappar, ljud, textfiler och andra resurser från sin egen domän. Dessutom tillåts alltid korsskriptning mellan två SWF-filer från samma domän, så länge båda filerna är skrivna i ActionScript 3.0. *Korsskriptning* innebär att kod i en fil kan komma åt de egenskaper, metoder och objekt som definieras av koden i en annan fil.

Korsskriptning används inte mellan SWF-filer skrivna i ActionScript 3.0 eller tidigare; däremot kan dessa filer kommunicera via klassen LocalConnection. Möjligheten för en SWF-fil att korsskripta SWF-filer i ActionScript 3.0 från andra domäner och att läsa in data från andra domäner är som standard otillåten. Tillstånd kan emellertid ges med ett anrop till metoden `Security.allowDomain()` i den inlästa SWF-filen. Mer information finns i ”Korsskriptning” på sidan 1067.

Följande grundläggande säkerhetsregler gäller alltid som standard:

- Resurser i samma säkerhetssandlåda har alltid åtkomst i båda riktningar.
- Körbara koder i filer i en fjärrsandlåda kan aldrig få åtkomst till lokala filer och data.

I körtiderna i Flash Player och AIR betraktas följande domäner som individuella, och tilldelas individuella säkerhetssandlådor:

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

Om en namngiven domän, till exempel http://example.com, har en mappning till en specifik IP-adress, till exempel http://192.0.34.166, tilldelas de separata säkerhetssandlådor i körtiderna.

Det finns två grundläggande metoder som en utvecklare kan använda för att ge SWF-filen åtkomst till resurser från andra sandlådor än den egna:

- Metoden `Security.allowDomain()` (se ”Författarinställningar (för utvecklare)” på sidan 1059)
- URL-principfilen (se ”Webbplatsinställningar (principfiler)” på sidan 1056)

I säkerhetsmodellerna för körtider i Flash Player och AIR är det en skillnad mellan inläsning av innehåll och extrahering eller åtkomst av data. *Innehåll* definieras som media, även visuella media som kan visas med körtider, ljud, video eller SWF-filer eller HTML som innehåller visningsmedier. *Data* definieras som något som är tillgängligt endast för kod. Innehåll och data läses in på olika sätt.

- Läsa in innehåll – du kan läsa in innehåll med hjälp av klasser som Loader, Sound och NetStream, med MXML-taggar när du använder Flex eller med HTML-taggar i ett AIR-program.
- Extrahera data – Du kan extrahera data från inläst medieinnehåll med hjälp av Bitmap-objekt, metoderna `BitmapData.draw()` och `BitmapData.drawWithQuality()`, egenskapen `Sound.id3` eller med metoden `SoundMixer.computeSpectrum()`. Metoden `drawWithQuality` är tillgänglig i Flash Player 11.3 och senare; AIR 3.3 och senare.
- Komma åt data – du kan komma åt data direkt genom att läsa in dem från en extern fil (till exempel en XML-fil) med hjälp av klasser som URLStream, URLLoader, FileReference, Socket och XMLSocket. I AIR tillhandahålls ytterligare klasser för att läsa in data, till exempel FileStream och XMLHttpRequest.

I Flash Players säkerhetsmodell definieras olika regler för inläsning av innehåll och åtkomst av data. Generellt finns det färre restriktioner för inläsning av innehåll än för åtkomst av data.

Generellt kan innehåll (SWF-filer, bitmappar, mp3-filer och videoklipp) läsas in från valfri plats, men om innehållet kommer från en annan domän än den där koden eller innehållet läses in, kommer det att partitioneras i en separat säkerhetssandlåda.

Det finns några begränsningar vid inläsning av innehåll:

- Som standard klassificeras lokala SWF-filer (som läses in från en icke-nätverksadress, t.ex. användarens hårddisk) i sandlådan lokal-med-filsystem. Sådana filer kan inte läsa in innehåll från nätverket. Mer information finns i ”Lokala sandlådor” på sidan 1049.
- Servrar som används med RTMP (Real-Time Messaging Protocol) kan ge begränsad åtkomst till innehåll. Mer information finns i ”Innehåll som levereras via RTMP-servrar” på sidan 1067.

Om ett inläst medium är en bild, ljud eller video kan dess data (t.ex. pixeldata, ljuddata) bara ges åtkomst av en SWF-fil utanför säkerhetssandlådan om SWF-filens domän ingick i en URL-principfil i mediets ursprungliga domän. Mer information finns i ”Åtkomst av inlästa medier som data” på sidan 1070.

Andra former av inlästa data omfattar text eller XML-filer, som läses in med objektet URLLoader. Om du vill läsa in data från annan säkerhetssandlåda, måste även i detta fall tillstånd beviljas via en URL-principfil i ursprungsdomänen. Mer information finns i ”Använda URLLoader och URLStream” på sidan 1073.

Obs! *Det krävs aldrig principfiler för att kod som körs i AIR-programsandlådan ska läsa in fjärrinnehåll eller fjärrdata.*

Säkerhetssandlådor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Klientdatorer kan hämta individuella filer som innehåller kod, innehåll och data från ett antal källor, till exempel externa webbplatser, från ett lokalt filsystem eller från ett installerat AIR-program. I Flash Player- och AIR-körtider tilldelas kodfiler och andra resurser enskilt till säkerhetssandlådor utifrån deras ursprung när de läses in. Exempel på resurser är delade objekt, bitmappar, ljud, video och datafiler. I följande avsnitt beskrivs de regler som upprätthålls av körtiderna och som styr vad en kod eller ett innehåll som körs inom en given sandlåda kan komma åt.

Mer information om säkerhet i Flash Player finns i avsnittet om säkerhet på Player Developer Center på www.adobe.com/go/devnet_security_en.

Fjärrsandlådor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash Player- och AIR-körtiderna klassificeras resurser (även SWF-filer) från Internet i separata sandlådor som motsvarar deras ursprungsdomän. Resurser som läses in från *example.com* placeras till exempel i en annan säkerhetssandlåda än resurser som läses in från *foo.org*. Som standard har dessa filer åtkomst till alla resurser från den egna servern. Externa SWF-filer kan ges åtkomst till ytterligare data från andra domäner via specifika webbplats- och författarbehörigheter, t.ex. URL-principfiler och metoden `Security.allowDomain()`. Mer information finns i ”Webbplatsinställningar (principfiler)” på sidan 1056 och ”Författarinställningar (för utvecklare)” på sidan 1059.

Externa SWF-filer kan inte läsa in några lokala filer eller resurser.

Mer information om säkerhet i Flash Player finns i avsnittet om säkerhet på Player Developer Center på www.adobe.com/go/devnet_security_en.

Lokala sandlådor

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Lokala filer beskriver filer som refereras via protokollet `file:` eller en UNC-sökväg (Universal Naming Convention). Lokala SWF-filer placeras i en av fyra lokala sandlådor:

- Sandlådan lokal-med-filsystem – i Flash Player- och AIR-körtiderna placeras av säkerhetsskäl alla lokala filer i sandlådan lokal-med-filsystem som standard. Från den här sandlådan kan körbar kod läsa lokala filer (genom att till exempel använda klassen URLLoader), men koden kan inte kommunicera med nätverket på något sätt. Detta gör att användaren kan vara säker på att lokala data inte läcker ut från nätverket eller delas ut på annat sätt.
- Sandlådan lokal-med-nätverk – vid kompileringen av en SWF-fil kan du ange att den har nätverksåtkomst vid körning som lokal fil (se ”Inställning av sandlådetyp för lokala SWF-filer” på sidan 1052). Dessa filer placeras i sandlådan lokal-med-nätverk. SWF-filer som har tilldelats sandlådan lokal-med-nätverk förlorar åtkomsten till de lokala filerna. I gengäld får SWF-filerna åtkomst till data i nätverket. En SWF-fil som är lokal-med-nätverk kan däremot fortfarande inte läsa data från nätverket, såvida det inte finns behörighet för den åtgärden, via en URL-principfil eller anrop till metoden `Security.allowDomain()`. En sådan behörighet tilldelas genom att en URL-principfil ger behörighet till *alla* domäner genom att använda `<allow-access-from domain="*"/>` eller `Security.allowDomain("*")`. Mer information finns i ”Webbplatsinställningar (principfiler)” på sidan 1056 och ”Författarinställningar (för utvecklare)” på sidan 1059.

- Sandlådan lokal-tillförlitlig – lokala SWF-filer som är registrerade som tillförlitliga (av användare eller installationsprogram) placeras i sandlådan lokal-tillförlitlig. Systemadministratörer och användare har även möjlighet att omtilldela (flytta) en lokal SWF-fil till eller från sandlådan lokal-tillförlitlig baserat på säkerhetsfrågor (se ”Administrativa inställningar” på sidan 1053 och ”Användarinställningar” på sidan 1055). SWF-filer som kopplats till sandlådan lokal-tillförlitlig kan samverka med alla andra SWF-filer och läsa in data från alla platser (externa eller lokala).
- Sandlådan application i AIR – den här säkerhetssandlådan rymmer innehåll som har installerats med AIR-programmet som körs. Som standard kan koder som körs i AIR-programsandlådan utföra korsskriptning för kod från alla domäner. Men filer som finns utanför AIR-programsandlådan tillåts inte utföra korsskriptning av kod i programsandlådan. Som standard kan kod och innehåll i AIR-programsandlådan läsa in innehåll och data från alla domäner.

Kommunikation mellan sandlådorna lokal-med-nätverk och lokal-med-filsystem, liksom kommunikation mellan sandlådan lokal-med-filsystem och externa sandlådor, är förbjuden. Behörighet att tillåta sådan kommunikation kan inte ges via ett program som körs i Flash Player eller av en användare eller administratör.

För skriptning i någondera riktning mellan lokala HTML-filer och lokala SWF-filer (t.ex. med klassen ExternalInterface) krävs att både HTML-filen och SWF-filen finns i sandlådan lokal-tillförlitlig. Detta beror på att de lokala säkerhetsmodellerna för webbläsare skiljer sig från den lokala säkerhetsmodellen i Flash Player.

SWF-filer i sandlådan lokal-med-nätverk kan inte läsa in SWF-filer i sandlådan lokal-med-filsystem. SWF-filer i sandlådan lokal-med-filsystem kan inte läsa in SWF-filer i sandlådan lokal-med-nätverk.

AIR-programsandlådan

Adobe AIR 1.0 och senare

I Adobe AIR-körningstiden läggs en extra sandlåda till, som kallas *program*sandlådan, i modellen för säkerhetssandlåda i Flash Player. De filer som installeras som en del av ett AIR-program läses in i programsandlådan. Alla andra filer som läses in av programmet har säkerhetsrestriktioner som motsvarar de som är angivna i den vanliga säkerhetsmodellen för Flash Player.

När ett program installeras, installeras alla filer som finns i ett AIR-paket till en programkatalog på användarens dator. Utvecklare kan använda URL-schemat `app:/` för att hänvisa till den här katalogen med kod (se ”URI-scheman” på sidan 815). Alla filer inuti programkatalogträdet tilldelas till programsandlådan när programmet körs. Innehåll i programsandlådan har de fullständiga behörigheter som är tillgängliga för ett AIR-program, inklusive interaktion med det lokala filsystemet.

Många AIR-program använder bara dessa lokalt installerade filer för att köra programmet. AIR-program är dock inte begränsade till bara filerna i programkatalogen - de kan läsa in alla typer av filer från alla källor. Detta inkluderar filer som är lokala på användarens dator och filer från tillgängliga fjärrkällor, som till exempel filer på ett lokalt nätverk eller på Internet. Filtyper har ingen påverkan på säkerhetsrestriktioner. Inlästa HTML-filer har samma säkerhetsbehörighet som inlästa SWF-filer från samma källa.

Innehåll i programsäkerhetssandlådan har tillgång till AIR-API:n som innehåll i andra sandlådor inte har tillgång till. Egenskapen `air.NativeApplication.nativeApplication.applicationDescriptor`, som returnerar innehållet i programbeskrivningsfilen för programmet, är till exempel begränsad till innehåll i programsäkerhetssandlådan. Ett annat exempel på ett begränsat API är klassen FileStream som innehåller metoder för att läsa och skriva i det lokala filsystemet.

ActionScript-API:er som endast är tillgängliga för innehåll i programsäkerhetssandlådan markeras med AIR-logotypen i *Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen*. Om dessa API:er används i andra sandlådor, initierar körtiden ett SecurityError-undantag.

För HTML-innehåll (i ett HTMLLoader-objekt) är alla AIR-JavaScript-API:n (de som är tillgängliga via egenskapen `window.runtime`, eller via objektet `air` när filen AIRAliases.js används) tillgängliga för innehåll i programsäkerhetssandlådan. HTML-innehåll i en annan sandlåda har inte åtkomst till egenskapen `window.runtime`, och därför kan inte innehållet komma åt API:erna för AIR eller Flash Player.

Följande ytterligare begränsningar gäller för innehållskörning inne i AIR-programmet:

- HTML-innehåll i programsäkerhetssandlådan har begränsningar i att använda API:er som kan omforma strängar dynamiskt till körbar kod efter att koden har lästs in. Detta är till för att förhindra programmet från att av misstag föra in (och köra) kod från källor som inte tillhör programmet (som till exempel potentiellt osäkra nätverksdomäner). Ett exempel är användningen av funktionen `eval()`. Detaljer finns under ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087.
- För att förhindra möjliga nätfiskeattacker ignoreras `img`-taggar i HTML-innehåll i ActionScript-objekt av typen TextField som ligger i programsäkerhetssandlådan.
- Innehåll i programsandlådan kan inte använda protokollet `asfunction` i HTML-innehåll i textfält av typen ActionScript 2.0.
- SWF-innehåll i programsandlådan kan inte använda korsdomän-cachen, vilket är en funktion som lades till i uppdatering 3 av Flash Player 9. Med den här funktionen kan Flash Player lagra Adobe-plattformskomponentinnehåll beständigt och återanvända det i SWF-innehåll vid behov (vilket eliminerar behovet att läsa in innehållet igen flera gånger).

Begränsningar för JavaScript inne i AIR

Adobe AIR 1.0 och senare

Till skillnad från innehåll i programsäkerhetssandlådan *kan* JavaScript-innehåll i en icke-programsäkerhetssandlåda anropa funktionen `eval()` när som helst för att köra dynamiskt genererad kod. Men det finns begränsningar för JavaScript som körs i en icke-programsäkerhetssandlåda inne i AIR. Dessa inkluderar:

- JavaScript-kod i en icke-programsandlåda har inte tillgång till objektet `window.runtime`, och kan därför inte köra AIR-API:er.
- Som standard kan innehåll i en icke-programsäkerhetssandlåda inte använda anrop av typen XMLHttpRequest för att läsa in data från andra domäner än den domän som gör anropet. Programkod kan dock skapa sådan åtkomst för icke-programinnehåll genom att ställa in ett `allowCrossdomainXHR`-attribut i innehållande frame eller iframe. Mer information finns i ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087.
- Det finns begränsningar för anrop till JavaScript-metoden `window.open()`. Mer detaljer finns under ”Begränsningar för anrop till JavaScript-metoden window.open()” på sidan 1090.
- HTML-innehåll i fjärrsäkerhetssandlådor (nätverk) kan bara läsa in CSS-, `frame`-, `iframe`- och `img`-innehåll från fjärrdomäner (nätverks-URL:er).
- HTML-innehåll i sandlådor av typerna lokal-med-filsystem, lokal-med-nätverk eller lokal-tillförlitlig, kan endast läsa in CSS-, `frame`-, `iframe`- och `img`-innehåll från lokala sandlådor (inte från program- eller nätverks-URL:er).

Detaljer finns under ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087.

Inställning av sandlådetyp för lokala SWF-filer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En användare av eller administratören för datorn kan ange att en lokal SWF-fil är tillförlitlig, vilket ger tillstånd att läsa in data från alla domäner, både lokala och i nätverk. Detta anges i katalogerna Global Flash Player Trust och User Flash Player Trust. Mer information finns i avsnitten ”Administrativa inställningar” på sidan 1053 och ”Användarinställningar” på sidan 1055.

Mer information om lokala sandlådor finns i ”Lokala sandlådor” på sidan 1049.

Adobe Flash Professional

Du kan konfigurera en SWF-fil för sandlådan lokal-med-filsystem eller lokal-med-nätverk genom att ange dokumentets publiceringsinställning i utvecklingsverktyget.

Adobe Flex

Du kan konfigurera en SWF-fil för sandlådan lokal-med-filsystem eller lokal-med-nätverk genom att ange flaggan `använd-nätverk` i Adobe Flex-kompilatorn. Mer information finns i ”Om alternativ för programkompilering” i *Bygga och installera Flex 3-program*.

Egenskapen Security.sandboxType

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En författare till en SWF-fil kan använda den skrivskyddade statiska egenskapen `Security.sandboxType` för att avgöra vilken sandlådetyp som SWF-filen har tilldelats i Flash Player eller AIR-körningstiden. Klassen Security innehåller följande konstanter som representerar tillåtna värden för egenskapen `Security.sandboxType`:

- `Security.REMOTE` – SWF-filen finns på en URL-adress och fungerar enligt domänbaserade sandlåderegler.
- `Security.LOCAL_WITH_FILE` – SWF-filen är en lokal fil som inte har registrerats som tillförlitlig av användaren och som inte har publicerats med en nätverksbeteckning. SWF-filen kan läsa från lokala datakällor, men inte kommunicera med Internet.
- `Security.LOCAL_WITH_NETWORK` – SWF-filen är en lokal fil som inte har registrerats som tillförlitlig av användaren, men som har publicerats med någon nätverksbeteckning. SWF-filen kan kommunicera med Internet men kan inte läsa från lokala datakällor.
- `Security.LOCAL_TRUSTED` – SWF-filen är en lokal fil som har registrerats som tillförlitlig av användaren, antingen med inställningshanteraren eller med en Trust-konfigurationsfil i Flash Player. SWF-filen kan både läsa från lokala datakällor och kommunicera med Internet.
- `Security.APPLICATION`) – SWF-filen körs i ett AIR-program och den installerades med paketet (AIR-filen) för det programmet. Som standard kan filer i sandlådan application i AIR utföra korsskriptning för valfri fil från alla domäner. Filer som finns utanför sandlådan application i AIR tillåts inte att utföra korsskriptning av AIR-filen. Som standard kan filer i sandlådan application i AIR läsa in innehåll och data från alla domäner.

Behörighetskontroller

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Säkerhetsmodellen för klientkörningen i Flash Player är utformad för resurser, vilka är objekt som t.ex. SWF-filer, lokala data och URL-adresser. Det är *intressenter* som äger eller använder dessa resurser. Intressenter kan ange säkerhetsinställningar för sina egna resurser och varje resurs har fyra intressenter. I Flash Player används en strikt behörighetshierarki för dessa inställningar:

Hierarki med säkerhetsinställningar

Detta innebär t.ex. att om en administratör begränsar åtkomsten till en resurs kan ingen annan intressent åsidosätta den begränsningen.

För AIR-program gäller de här behörighetskontrollerna endast innehåll som körs utanför AIR-programsandlådan.

Administrativa inställningar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En administratörsanvändare (inloggad med administratörsbehörigheter) kan tillämpa säkerhetsinställningar i Flash Player som påverkar alla andra användare. I en ickeföretagsmiljö, t.ex. i en dator i hemmet, är det vanligtvis en användare som även har administratörsbehörighet. Även i en företagsmiljö kan individuella användare ha administratörsbehörighet till datorn.

Det finns två typer av administrativa användarinställningar:

- Filen mms.cfg
- Katalogen Global Flash Player Trust

Filen mms.cfg

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Filen mms.cfg är en textfil med vilken administratörer kan bevilja eller begränsa åtkomsten till olika funktioner. När Flash Player startas läses säkerhetsinställningarna in från den här filen och används för att begränsa funktioner. Filen mms.cfg innehåller inställningar som administratören använder för att hantera funktioner som sekretessinställningar, lokal filsäkerhet, socketanslutningar och så vidare.

En SWF-fil har åtkomst till viss information om funktioner som har avaktiverats via anrop av egenskaperna `Capabilities.avHardwareDisable` och `Capabilities.localFileReadDisable`. Däremot är de flesta inställningarna i filen mms.cfg oåtkomliga från ActionScript.

Om du vill tillämpa programoberoende säkerhets- och sekretesspolicy för en dator ska mms.cfg-filen ändras av enbart systemadministratörer. Filen mms.cfg är inte avsedd att användas av installationsprogram. Trots att ett installationsprogram som körs med administratörsbehörighet kan ändra innehållet i filen mms.cfg betraktar Adobe en sådan ändring som överträdelse från användarens sida och uppmanar utvecklare av installationsprogram att aldrig ändra filen mms.cfg.

Filen mms.cfg sparas på följande plats:

- Windows: *system*\Macromed\Flash\mms.cfg

  (t.ex. C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac: *app support*/Macromedia/mms.cfg

  (t.ex. /Library/Application Support/Macromedia/mms.cfg)

Mer information om filen mms.cfg finns i administrationshandboken för Flash på www.adobe.com/go/flash_player_admin_se.

Katalogen Global Flash Player Trust

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Administratörsanvändare och installationsprogram kan registrera angivna lokala SWF-filer som tillförlitliga för alla användare. Dessa SWF-filer är tilldelade sandlådan lokal-tillförlitlig. De kan användas med valfria andra SWF-filer och de kan läsa in data från valfri annan plats; externt eller lokalt. Filer betecknas som tillförlitliga i katalogen Global Flash Player Trust, på följande plats:

- Windows: *system*\Macromed\Flash\FlashPlayerTrust

  (t.ex. C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac: *app support*/Macromedia/FlashPlayerTrust

  (t.ex. /Library/Application Support/Macromedia/FlashPlayerTrust)

Katalogen Flash Player Trust kan innehålla valfritt antal textfiler, där var och en innehåller tillförlitliga sökvägar, med en sökväg per rad. Varje sökväg kan vara en enskild SWF-fil, HTML-fil eller katalog. Kommentarsrader inleds med symbolen `#`. En Trust-konfigurationsfil i Flash Player med följande text ger tillförlitlighetsstatus till alla filer i angiven katalog och alla underkataloger:

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

Sökvägarna som anges i en Trust-konfigurationsfil måste alltid vara lokala sökvägar eller sökvägar i SMB-nätverk. Eventuella HTTP-sökvägar i en Trust-konfigurationsfil ignoreras; endast lokala filer kan vara tillförlitliga.

För att undvika konflikter ger du varje Trust-konfigurationsfil ett filnamn som motsvarar installationsprogrammet och använder .cfg som filtillägg.

Som utvecklare som fördelar en lokal SWF-fil via ett installationsprogram kan du se till att installationsprogrammet lägger till en konfigurationsfil till katalogen Global Flash Player Trust. På så vis tilldelas fullständig behörighet till filen du fördelar till användarna. Installationsprogrammet måste köras av en användare med administratörsbehörighet. Till skillnad från filen mms.cfg inkluderas katalogen Global Flash Player Trust för att installationsprogram ska tilldela behörighet för tillförlitlighet. Både administratörsanvändare och installationsprogram kan ange tillförlitliga lokala program med hjälp av katalogen Global Flash Player Trust.

Det finns även Flash Player Trust-kataloger för individuella användare (se ”Användarinställningar” på sidan 1055).

Användarinställningar

Flash Player 9 och senare

Flash Player innehåller tre olika mekanismer för behörighetsinställningar på användarnivå: inställningsgränssnittet, inställningshanteraren och katalogen User Flash Player Trust.

Inställningsgränssnittet och inställningshanteraren

Flash Player 9 och senare

Inställningsgränssnittet är en snabb, interaktiv mekanism för inställningskonfigurationen för en specifik domän. Inställningshanteraren har ett mer detaljerat gränssnitt och ger möjlighet till globala ändringar som påverkar behörigheter för flera eller alla domäner. Dessutom, vid en begäran om en ny behörighet av en SWF-fil där beslut avseende säkerhet eller sekretess krävs under körning, så visas dialogrutor där användare kan justera vissa inställningar i Flash Player.

Inställningshanteraren och inställningsgränssnittet innehåller säkerhetsalternativ, som kamera- och mikrofoninställningar, inställningar för lagring av delade objekt, inställningar för äldre innehåll och så vidare. Varken inställningshanteraren eller inställningsgränssnitten är tillgängliga för AIR-program.

Obs! *Eventuellt gjorda inställningar i filen mms.cfg (se ”Administrativa inställningar” på sidan 1053) visas inte i inställningshanteraren.*

Mer information om inställningshanteraren finns på www.adobe.com/go/settingsmanager_se.

Katalogen User Flash Player Trust

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Användare och installationsprogram kan registrera angivna lokala SWF-filer som tillförlitliga. Dessa SWF-filer är tilldelade sandlådan lokal-tillförlitlig. De kan användas med valfria andra SWF-filer och de kan läsa in data från valfri annan plats; externt eller lokalt. En användare anger en fil som tillförlitlig i katalogen User Player Trust, som finns i samma katalog som lagringsutrymmet för delade objekt, på följande platser (platserna är specifika för aktuell användare):

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (t. ex. C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust i Windows XP eller C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust i Windows Vista)

  I Windows döljs mappen Application Data som standard. Om du vill visa dolda mappar och filer öppnar du Utforskaren i Windows med Den här datorn, väljer Verktyg > Mappalternativ och markerar sedan fliken Visa. Markera alternativet Visa dolda filer och mappar på fliken Visa.

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (t.ex. /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  Dessa inställningar påverkar endast den aktuella användaren, inga andra som loggar in på datorn. Om en användare utan administratörsbehörighet installerar ett program i sin egen del i systemet så registreras programmet, via katalogen User Flash Player Trust, som tillförlitligt för just den användaren.

  Som utvecklare som fördelar en lokal SWF-fil via ett installationsprogram kan du se till att installationsprogrammet lägger till en konfigurationsfil till katalogen User Flash Player Trust. På så vis tilldelas fullständig behörighet till filen du fördelar till användarna. Även i den här situationen utförs en kontroll av användarinställningar i filen i User Flash Player Trust, eftersom det är en användaråtgärd som initierar den.

  Det finns även en Global Flash Player Trust-katalog som används av administratören eller installationsprogram vid registrering av program för samtliga användare i ett datorsystem (se ”Administrativa inställningar” på sidan 1053).

Webbplatsinställningar (principfiler)

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att göra data från en webbserver tillgängliga för SWF-filer från andra domäner kan du skapa en principfil på servern. En *principfil* är en XML-fil som placeras på en särskild plats på servern.

Principfiler påverkar åtkomsten till flera resurser, exempelvis:

- Data i bitmappar, ljud och video
- Inläsning av XML- och textfiler
- Import av SWF-filer från andra säkerhetsdomäner till säkerhetsdomänen för inläsande SWF-fil
- Åtkomst till socket- och XML-socketanslutningar

ActionScript-objekt instansierar två olika typer av serveranslutningar: dokumentbaserade serveranslutningar och socketanslutningar. ActionScript-objekt som t.ex. Loader, Sound, URLLoader och URLStream instansierar dokumentbaserade serveranslutningar och var och en av dessa objekt läser in en fil från en URL-adress. Objekt i ActionScript Socket och XMLSocket gör socketanslutningar som används med direktuppspelningsdata, inte inlästa dokument.

Eftersom Flash Player har stöd för två typer av serveranslutningar, finns det två typer av principfiler: URL-principfiler och socketprincipfiler.

- För dokumentbaserade anslutningar krävs *URL-principfiler*. Med dessa filer kan servern ange att dess data och dokument är tillgängliga för SWF-filer som hanteras från vissa domäner eller från alla domäner.
- För socketanslutningar krävs *socketprincipfiler*, som möjliggör nätverksanslutning direkt vid en lägre TCP-socketnivå, med hjälp av klasserna Socket och XMLSocket.

Flash Player kräver att principfilerna överförs med samma protokoll som anslutningen vill använda. Om du till exempel placerar en principfil på HTTP-servern ges SWF-filer från andra domäner tillstånd att läsa in data från den som HTTP-server. Om du inte anger någon socketprincipfil på samma server tillåter du inte SWF-filer från andra domäner att ansluta till servern på socketnivå. Metoden som används vid hämtning av socketprincipfilen måste m.a.o. matcha anslutningsmetoden.

Hur principfiler används och syntaxen beskrivs i korthet i resten av avsnittet, då de gäller för SWF-filer som publiceras för Flash Player 10. (Hur principfiler implementeras skiljer sig något från tidigare versioner av Flash Player, eftersom säkerheten har förstärkts i senare versioner av Flash Player.) Mer information om principfiler finns i avsnittet om ändringar av principfiler i Flash Player 9 på Flash Player Developer Center på www.adobe.com/go/devnet_security_en.

Kod som körs i AIR-programsandlådan kräver ingen principfil för att komma åt data från en URL eller socket. Kod i ett AIR-program som körs i en icke-programsandlåda kräver ingen principfil.

Huvudprincipfiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player (och AIR-innehåll som inte finns i AIR-sandlådan application) söker som standard först efter en URL-principfil med namnet `crossdomain.xml` i rotkatalogen på servern, och söker sedan efter en socketprincipfil på port 843. En fil som finns på någon av dessa platser kallas för *huvudprincipfil*. (När det gäller socketanslutningar söker Flash Player också efter en socketprincipfil på samma port som huvudanslutningen. En principfil som finns vid den porten betraktas emellertid inte som någon huvudprincipfil.)

Förutom att ange åtkomstbehörigheter kan huvudprincipfilen också innehålla en *metaprincip*-sats. En metaprincip anger vilka platser som kan innehålla principfiler. Standardmetaprincipen för URL-principfiler är ”master-only”, vilket innebär att /crossdomain.xml är den enda principfil som tillåts på servern. Standardmetaprincipen för socketprincipfiler är ”all”, vilket innebär att alla socketar på värden kan fungera som socketprincipfil.

Obs! *I Flash Player 9 och tidigare var standardprinciperna för URL-principer ”all”, vilket innebär att alla kataloger kan innehålla en principfil. Om du har installerat program som läser in principfiler från andra platser än standardfilen /crossdomain.xml, och dessa program kanske inte körs i Flash Player 10, måste du (eller serveradministratören) ändra huvudprincipfilen så att andra principfiler tillåts. Mer information om hur du anger en annan metaprincip finns i avsnittet om ändringar av principfiler i Flash Player 9 på Flash Player Developer Center på www.adobe.com/go/devnet_security_en.*

En SWF-fil kan söka efter ett annat principfilsnamn eller en annan katalog genom anrop av metoden `Security.loadPolicyFile()`. Om huvudprincipfilen inte anger att målplatsen kan hantera principfiler, har anropet till `loadPolicyFile()` ingen effekt, även om det finns en principfil på den platsen. Anropa `loadPolicyFile()` innan du försöker med några nätverksåtgärder som kräver principfilen. Flash Player ställer automatiskt nätverksförfrågningar i kö bakom deras motsvarande principfilsförsök. Det går till exempel att anropa `Security.loadPolicyFile()` omedelbart innan en nätverksåtgärd initieras.

När Flash Player söker efter en huvudprincipfil väntar programmet i tre sekunder på ett svar från servern. Om inget svar kommer, förutsätts det att det inte finns någon huvudprincipfil. Det finns emellertid inget standardvärde för timeout för anrop till `loadPolicyFile()`. Flash Player förutsätter att filen som anropas finns, och väntar så länge det behövs för att läsa in den. Om du vill vara säker på att huvudprincipfilen läses in ska du därför anropa den explicit med `loadPolicyFile()`.

Även om metoden heter `Security.loadPolicyFile()`, läses ingen principfil in förrän ett nätverksanrop som kräver en principfil skickas. Anrop till `loadPolicyFile()` anger bara för Flash Player var programmet ska söka efter principfiler när de behövs.

Du kan inte meddelas om när en principfilsbegäran initieras eller slutförs, och det finns ingen anledning till att detta ska göras. Flash Player utför principkontroller asynkront, och väntar med att initiera anslutningarna tills principfilskontrollerna har slutförts.

Avsnitten nedan innehåller information som bara gäller för URL-principfiler. Mer information om socketprincipfiler finns i ”Ansluta till socketar” på sidan 1073.

URL-principfilens omfång

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En URL-principfil gäller bara för den katalog varifrån inläsningen görs och i respektive underkataloger. En principfil i rotkatalogen tillämpas alltså på hela servern, medan en principfil som läses in från någon underkatalog gäller endast i den katalogen och respektive underkataloger.

En principfil påverkar åtkomsten endast till den specifika servern där den finns placerad. En principfil som t.ex. finns på https://www.adobe.com:8080/crossdomain.xml tillämpas bara på datainläsningsanrop som görs till www.adobe.com över HTTPS på porten 8080.

Ange åtkomstbehörighet i en URL-principfil

Flash Player 9 och senare, Adobe AIR 1.0 och senare

En principfil innehåller en enstaka `<cross-domain-policy>`-tagg, vilken i sin tur innehåller noll eller flera `<allow-access-from>`-taggar. Varje `<allow-access-from>`-tagg innehåller ett attribut, `domain`, vilket anger antingen en exakt IP-adress, en exakt domän eller en jokerteckendomän (valfri domän). Jokerteckendomäner anges på något av två sätt:

- Men en enkel asterisk (*), som matchar alla domäner och alla IP-adresser
- Med en asterisk följt av ett suffix, som bara matchar de domäner som slutar med angivet suffix

Suffixen måste börja med en punkt. Jokerteckendomäner med suffix kan emellertid matcha domäner som består av bara suffixet utan inledande punkt. Till exempel betraktas xyz.com att tillhöra *.xyz.com. Jokertecken tillåts inte i IP-domänspecifikationer.

I följande exempel visas en URL-principfil som ger åtkomst till SWF-filer som kommer från *.example.com, www.friendOfExample.com och 192.0.34.166:

```
<?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>
```

Om du anger en IP-adress ges åtkomst bara till SWF-filer som lästs in från denna IP-adress med IP-syntax (t.ex. http://65.57.83.12/flashmovie.swf). Ingen åtkomst beviljas till SWF-filer som använder domännamnssyntax. DNS-upplösning utförs inte av Flash Player.

Du kan ge åtkomst till dokument som härstammar från valfri domän, vilket visas i följande exempel:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Varje `<allow-access-from>`-tagg har även ett valfritt `secure`-attribut, vars standardvärde är `true`. Du kan ange `false` för attributet om din principfil finns på en HTTPS-server och om du vill tillåta SWF-filer på en icke-HTTP-server att läsa in data från HTTPS-servern.

Om `secure`-attributet har värdet `false` kan säkerheten som HTTPS erbjuder äventyras. Speciellt om attributet anges med `false`, då öppnas säkert innehåll för spionage och bedrägeri. Adobe rekommenderar att du inte anger attributet `secure` med `false`.

Om de data som ska läsas in finns på en HTTPS-server, men SWF-filen som läser in dem finns på en HTTP-server, bör du flytta den inläsande SWF-filen till en HTTPS-server. På så sätt skyddas alla data med HTTPS. Om du däremot beslutar att du måste behålla inläsande SWF-fil på en HTTP-server så lägger du till attributet `secure="false"` till `<allow-access-from>`-taggen, enligt följande kod:

```
<allow-access-from domain="www.example.com" secure="false" />
```

Ett annat element som du kan använda för att bevilja åtkomst är taggen `allow-http-request-headers-from`. Det här elementet beviljar en klient som är värd för innehåll från en annan behörighetsdomän att skicka användardefinierade huvuden till din domän. Taggen `<allow-access-from>` ger andra domäner behörighet att hämta data från din domän, medan taggen `allow-http-request-headers-from` ger andra domäner behörighet att skicka data till din domän, i form av huvuden. I följande exempel tillåts alla domäner att skicka SOAPAction-huvudet till den aktuella domänen:

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Om satsen `allow-http-request-headers-from` finns i huvudprincipfilen, gäller den för alla kataloger på värden. Annars gäller den bara för katalogen och underkatalogerna för principfilen som innehåller programsatsen.

Förhandsladda principfiler

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Att läsa in data från en server eller ansluta till en socket är en asynkron åtgärd. Flash Player väntar på att principfilen ska sluta laddas ned innan huvudåtgärden inleds. Att extrahera pixeldata från bilder eller extrahera exempeldata från ljud är emellertid synkrona åtgärder. Principfilen måste läsas in innan du kan extrahera data. När du läser in media anger du att det ska söka efter en principfil:

- När du använder metoden `Loader.load()` anger du egenskapen `checkPolicyFile` för parametern `context`, vilket är ett LoaderContext-objekt.
- När du bäddar in en bild i ett textfält med hjälp av `<img>`-taggen anger du attributet `checkPolicyFile` för `<img>`-taggen med `"true"` enligt följande:

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- När du använder metoden `Sound.load()` anger du egenskapen `checkPolicyFile` för parametern `context`, vilket är ett SoundLoaderContext-objekt.
- När du använder klassen NetStream anger du egenskapen `checkPolicyFile` för NetStream-objektet.

När du anger en av dessa parametrar söker Flash Player först efter principfiler som redan har laddats ned för domänen. Sedan söker programmet efter principfilen på standardplatsen på servern, och söker både efter programsatsen `<allow-access-from>` och en metaprincip. Slutligen undersöks om väntande anrop till metoden `Security.loadPolicyFile()` finns i omfånget.

Författarinställningar (för utvecklare)

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Det huvud-API för ActionScript som används för att tilldela behörighet är metoden `Security.allowDomain()`. Den ger behörighet till SWF-filer i de domäner du anger. I följande exempel tilldelar en SWF-fil åtkomst till SWF-filer som hanteras från domänen www.example.com:

```
Security.allowDomain("www.example.com")
```

Den här metoden ger behörighet till följande:

- Korsskriptning mellan SWF-filer (se ”Korsskriptning” på sidan 1067)
- Åtkomst av visningslista (se ”Gå igenom visningslistan” på sidan 1069)
- Händelseavkänning (se ”Säkerhet med händelser” på sidan 1070)
- Fullständig åtkomst till egenskaper och metoder i Stage-objektet (se ”Scen, säkerhet” på sidan 1068)

Det primära syftet med anrop av metoden `Security.allowDomain()` är att tilldela behörighet till SWF-filer i en extern domän som ska skripta SWF-filen som anropar metoden `Security.allowDomain()`. Mer information finns i ”Korsskriptning” på sidan 1067.

Att en IP-adress anges som parameter till metoden `Security.allowDomain()` innebär inte att alla parter från den angivna IP-adressen får åtkomst. Endast de parter som lästes in genom att IP-adressen angavs i deras webbadresser ges åtkomst, i stället för med ett domännamn som mappar till IP-adressen. Om till exempel domännamnet www.example.com mappar till IP-adressen 192.0.34.166 kommer ett anrop till `Security.allowDomain("192.0.34.166")` inte att ges åtkomst till www.example.com.

Du kan skicka jokertecknet `"*"` till metoden `Security.allowDomain()` för att tillåta åtkomst från alla domäner. Eftersom det ger behörighet för SWF-filer från *alla* domäner att skripta anropande SWF-fil måste du använda jokertecknet `"*"` med försiktighet.

ActionScript innehåller ett andra behörighets-API, med namnet `Security.allowInsecureDomain()`. Den här metoden gör samma sak som metoden `Security.allowDomain()`, förutom att den vid anrop från en SWF-fil som opereras från en säker HTTPS-anslutning dessutom ger åtkomst till anropande SWF-fil via andra SWF-filer som opereras från ett osäkert protokoll (t.ex. HTTP). Däremot är det ingen bra säkerhetsrutin att tillåta skriptning mellan filer från ett säkert protokoll (HTTPS) och från osäkra protokoll (HTTP); detta öppnar säkert innehåll för spionage och bedrägeri. Så här kan ett angrepp se ut: eftersom metoden `Security.allowInsecureDomain()` tillåter åtkomst till dina säkra HTTPS-data via SWF-filer som opereras över HTTP-anslutningar, kan en angripare som gått emellan din HTTP-server och dina användare ersätta HTTP SWF-filen med sin egen, vilket då kan ge åtkomst till dina HTTPS-data.

Viktigt! *Kod som körs i AIR-programsandlådan har inte behörighet att anropa metoderna `allowDomain()` eller `allowInsecureDomain()` i klassen Security.*

En annan viktig säkerhetsrelaterad metod är `Security.loadPolicyFile()`, som gör att Flash Player söker efter en principfil på en icke-standardplats. Mer information finns i ”Webbplatsinställningar (principfiler)” på sidan 1056.

Begränsa nätverks-API:er

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Nätverks-API:er kan begränsas på två sätt. Skadlig aktivitet förhindras genom att åtkomsten till portar som brukar reserveras blockeras. Du kan inte åsidosätta dessa spärrar i koden. Om du vill styra en SWF-fils åtkomst till nätverksfunktioner med hänsyn till andra portar använder du inställningen `allowNetworking`.

Blockerade portar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player och Adobe AIR (och även webbläsare) har bara HTTP-åtkomst via vissa portar. HTTP-förfrågningar tillåts inte på vissa standardportar som brukar användas för andra typer av servrar än HTTP-servrar.

Dessas portbegränsningar gäller för alla API:er som försöker komma åt en nätverks-URL. Det enda undantaget är API:er som anropar socketar direkt, till exempel `Socket.connect()` och `XMLSocket.connect()`, eller anrop till `Security.loadPolicyFile()` där en socketprincipfil läses in. Socketanslutningar tillåts eller nekas via socketprincipfiler på målservern.

I listan nedan visas vilka ActionScript 3.0-API:er som portblockering gäller för:

`FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `Loader.loadBytes()`, `navigateToURL()`, `NetConnection.call()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `sendToURL()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`

Portblockering gäller också för import av delade bibliotek, användningen av taggen `<img>` i textfält och inläsningen av SWF-filer till en HTML-sida med hjälp av taggarna `<object>` och `<embed>`.

Portblockering gäller också för användningen av taggen `<img>` i textfält och inläsningen av SWF-filer till en HTML-sida med hjälp av taggarna `<object>` och `<embed>`.

I listan nedan visas vilka portar som blockeras:

HTTP: 20 (ftp data), 21 (ftp control)

HTTP och FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

Använda parametern allowNetworking

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan kontrollera SWF-filens åtkomst till nätverksfunktioner genom inställningar i `allowNetworking`-parametern i `<object>`- och `<embed>`-taggarna på HTML-sidan där SWF-innehållet finns.

Tillåtna värden för `allowNetworking`:

- `"all"` (standardvärde) – Alla nätverks-API:er tillåts i SWF-filen.
- `"internal"` – SWF-filen får inte anropa webbläsarnavigering eller webbläsarinteraktioner för API:er (se längre fram i avsnittet), men den kan anropa alla andra nätverks-API:er.
- `"none"` – SWF-filen får inte anropa webbläsarnavigering eller webbläsarinteraktioner för API:er (se längre fram i avsnittet) och den kan inte använda några kommunikations-API:er av typen SWF-till-SWF (också längre fram i avsnittet).

Parametern `allowNetworking` används i huvudsak när SWF-filen och den omslutande HTML-sidan kommer från olika domäner. Du bör inte använda värdet `"internal"` eller `"none"` när SWF-filen läses in från samma domän som de omslutande HTML-sidorna, eftersom du inte kan säkerställa att en SWF-fil alltid läses in med rätt HTML-sida. Otillförlitliga parter kan läsa in en SWF-fil från din domän utan omslutande HTML, och då fungerar inte begränsningen `allowNetworking` på det sätt som du har tänkt.

Om en icke tillgänglig API anropas genereras ett SecurityError-undantag.

Lägg till parametern `allowNetworking` och ange dess värde i taggarna `<object>` och `<embed>` på HTML-sidan som innehåller en referens till SWF-filen, enligt följande exempel:

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code
base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,
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>
```

En HTML-sida kan också använda ett skript för att generera SWF-inbäddningstaggar. Du måste ändra skriptet så att det lägger in rätt `allowNetworking`-inställningar. HTML-sidor som genererats av Adobe Flash Professional och Adobe Flash Builder använder funktionen `AC_FL_RunContent()` för att bädda in referenser till SWF-filer. Lägg till inställningarna för parametern `allowNetworking` i skriptet enligt följande:

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Följande API:er är inte tillgängliga när `allowNetworking` har värdet `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

Förutom API:erna i ovanstående lista är följande API:er inte heller tillgängliga när `allowNetworking` har värdet `"none"`:

`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()`

Även om den markerade `allowNetworking`-inställningen tillåter en SWF-fil att använda ett nätverks-API kan det finnas andra restriktioner på grund av säkerhetssandlådans begränsningar (se ”Säkerhetssandlådor” på sidan 1049).

Om `allowNetworking` anges till `"none"` kan du inte referera till externa medier i en `<img>`-tagg i egenskapen `htmlText` i ett TextField-objekt (ett SecurityError-undantag sker).

När `allowNetworking` är inställt på `none` blockeras en symbol från ett importerat delat bibliotek som lagts till i Flash Professional (inte ActionScript) vid körning.

Säkerhet med helskärmsläge

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash Player 9.0.27.0 och senare finns helskärmsläge, där innehållet som körs i Flash Player kan fylla hela skärmen. För att aktivera helskärmsläget anges egenskapen `displayState` för scenen med konstanten `StageDisplayState.FULL_SCREEN`. Mer information finns i avsnittet ”Arbeta i helskärmsläge” på sidan 167.

För SWF-filer som körs i en fjärrsandlåda finns det vissa säkerhetsfrågor du bör tänka på.

Aktivera helskärmsläge genom att i taggarna `<object>` och `<embed>` på HTML-sidan som innehåller en referens till SWF-filen lägga till parametern `allowFullScreen` med värdet `"true"` (standardvärdet är `"false"`), som visas i följande exempel:

```
<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>
```

En HTML-sida kan också använda ett skript för att generera SWF-inbäddningstaggar. Du måste ändra skriptet så att det lägger in rätt `allowFullScreen`-inställningar. HTML-sidor som genererats av Flash Professional och Flash Builder använder funktionen `AC_FL_RunContent()` för att bädda in referenser till SWF-filer, och du måste lägga till inställningarna för parametern `allowFullScreen` enligt följande:

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

ActionScript som initierar helskärmsläge kan anropas endast i anslutning till en mus- eller tangentbordshändelse. Om det anropas i andra situationer genereras ett undantag i Flash Player.

Ett meddelande visas när innehållet kommer att visas i helskärmsläge. Användaren får instruktioner om hur man avbryter och återgår till normalläge. Meddelandet visas under några sekunder och tonas sedan bort.

För innehåll som körs i en webbläsare är tangentbordsanvändningen begränsad i helskärmsläge. I Flash Player 9 stöds bara kortkommandon som återför programmet till normalläge, till exempel Esc. Användarna kan inte skriva text i textfälten eller navigera på skärmen. I Flash Player 10 och senare stöds icke utskrivbara tangenter (till exempel pilar, blanksteg och tabb). Det går emellertid fortfarande inte att skriva text.

Helskärmsläge tillåts alltid i den fristående spelaren eller i en projektorfil. Tangentbordsanvändning (inklusive textskrivning) stöds helt och hållet i dessa miljöer.

Anrop av `displayState`-egenskapen för ett Stage-objekt ger ett undantag för varje anropare som inte befinner sig i samma säkerhetssandlåda som Stageägaren (SWF-huvudfilen). Mer information finns i ”Scen, säkerhet” på sidan 1068.

Administratörer kan avaktivera helskärmsläget för SWF-filer som körs i webbläsare genom inställning av `FullScreenDisable = 1` i filen mms.cfg. Mer information finns i ”Administrativa inställningar” på sidan 1053.

I en webbläsare måste SWF-filen finnas med i en HTML-sida för att tillåta helskärmsläge.

Säkerhet för interaktivt helskärmsläge

Flash Player 11.3 och senare, Adobe AIR 1.0 och senare

Flash Player 11.3 och senare versioner har stöd för interaktivt helskärmsläge, i vilket innehåll som körs i Flash Player kan fylla hela skärmen *och ta emot textinmatningar*. När interaktivt helskärmsläge ska användas anges egenskapen `displayState` för scenen till konstanten `StageDisplayState.FULL_SCREEN_INTERACTIVE`. Mer information finns i avsnittet ”Arbeta i helskärmsläge” på sidan 167.

För SWF-filer som körs i en fjärrsandlåda finns det vissa säkerhetsfrågor du bör tänka på.

Om du vill aktivera helskärmsläge lägger du till parametern `allowFullScreenInteractive` med dess värde inställt på `"true"` (standardvärdet är `"false"`) i taggarna `<object>` och `<embed>`) på den HTML-sida som innehåller en referens till SWF-filen, enligt nedanstående exempel:

```
<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="allowFullScreenInteractive" 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>
```

En HTML-sida kan också använda ett skript för att generera SWF-inbäddningstaggar. Du måste ändra skriptet så att rätt `allowFullScreenInteractive`-inställningar infogas. HTML-sidor som genererats av Flash Professional och Flash Builder använder funktionen `AC_FL_RunContent()` för att bädda in referenser till SWF-filer, och du måste lägga till inställningarna för parametern `allowFullScreenInteractive` enligt följande:

```
AC_FL_RunContent( ... "allowFullScreenInteractive", "true", ...)
```

Den ActionScript-kod som initierar det interaktiva helskärmsläget kan bara anropas som svar på en mushändelse eller tangentbordshändelse. Om det anropas i andra situationer genereras ett undantag i Flash Player.

Det visas ett meddelande när innehållet går över i interaktivt helskärmsläge. Meddelandet visar domänen för helskärmssidan, anvisningar om hur helskärmsläget avslutas och knappen **Tillåt**. Meddelandet ligger kvar tills användaren klickar på **Tillåt** och godtar det interaktiva helskärmsläget.

Administratörer kan inaktivera det interaktiva helskärmsläget för SWF-filer som körs i webbläsare genom att ange `FullScreenInteractiveDisable = 1` i filen mms.cfg. Mer information finns i ”Administrativa inställningar” på sidan 1053.

I en webbläsare måste en SWF-fil finnas på en HTML-sida för att interaktivt helskärmsläge ska kunna användas.

Inläsning av innehåll

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player- och AIR-innehåll kan läsa in många typer av annat innehåll, däribland följande:

- SWF-fil
- bilder
- Ljud
- Video
- HTML-filer (endast AIR)
- JavaScript (endast AIR)

Läsa in SWF-filer och bilder med klassen Loader

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda klassen Loader för att läsa in SWF-filer och bilder (JPG-, GIF- eller PNG-filer). Alla SWF-filer, förutom den i sandlådan lokal-med-filsystem, kan läsa in SWF-filer och bilder från valfri nätverksdomän. Endast SWF-filer i lokala sandlådor kan läsa in SWF-filer och bilder från det lokala filsystemet. Däremot kan filer i sandlådan lokal-med-nätverk endast läsa in lokala SWF-filer som finns i sandlådan lokal-tillförlitlig eller lokal-med-nätverk. SWF-filer i sandlådan lokal-med-nätverk läser in lokalt innehåll som inte är SWF-filer (t.ex. bilder), däremot har de ingen åtkomst till data i det inlästa innehållet.

Vid inläsning av en SWF-fil från en otillförlitlig källa (exempelvis en annan domän än Loader-objektets rot-SWF-fil) kanske du vill definiera en mask för Loader-objektet för att förhindra att det inlästa innehållet (som är underordnad Loader-objektet) ritar delar av scenen utanför den masken, som framgår av följande kod:

```
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);
```

När du anropar metoden `load()` i objektet Loader så kan du ange parametern `context`, vilket är ett LoaderContext-objekt. Klassen LoaderContext innehåller tre egenskaper som du anger för att definiera hur inläst innehåll ska användas:

- `checkPolicyFile`: Använd den här egenskapen endast när du läser in en bildfil (inte en SWF-fil). Ange egenskapen för en bildfil från en annan domän än den som filen med Loader-objektet har. Om du anger egenskapen med `true` kontrolleras om det finns en URL-principfil på den ursprungliga servern (se ”Webbplatsinställningar (principfiler)” på sidan 1056). Om servern ger behörighet till Loader-domänen så får ActionScript från SWF-filer i Loader-domänen åtkomst till data i inläst bild. Du kan alltså använda egenskapen `Loader.content` för att hämta en referens till det Bitmap-objekt som representerar den inlästa bilden eller metoderna `BitmapData.draw()` och `BitmapData.drawWithQuality()` för att komma åt pixlar från den inlästa bilden. Metoden `drawWithQuality` är tillgänglig i Flash Player 11.3 och senare; AIR 3.3 och senare.
- `securityDomain`: Använd bara den här egenskapen när du läser in en SWF-fil (inte en bild). Ange den här för en SWF-fil från en annan domän än den som filen med Loader-objektet har. Det finns endast två värden som för tillfället kan användas för egenskapen `securityDomain`: `null` (standard) och `SecurityDomain.currentDomain`. Om du anger `SecurityDomain.currentDomain` krävs det att inläst SWF-fil har *importerats* till sandlådan med inläsande SWF-fil, vilket innebär att den används som om den vore inläst från den inläsande SWF-filens egen server. Detta tillåts bara om en URL-principfil finns på den inlästa SWF-filens server, vilket ger åtkomst via den inläsande SWF-filens domän. Om principfilen som krävs finns kan inläsare och utläsare tillåtas skripta varandra när inläsningen påbörjas, eftersom de ligger i samma sandlåda. Observera, att sandlådeimport i de flesta fall kan ersättas med att du gör en vanlig inläsning och sedan ser till att inläst SWF-fil anropar metoden `Security.allowDomain()`. Den senare metoden kan vara enklare att använda, eftersom den inlästa SWF-filen då kommer att ligga i sin egen vanliga sandlåda och har därför åtkomst till resurser på den egna servern.

- `applicationDomain`: Använd bara den här egenskapen när du läser in en SWF-fil som är skriven i ActionScript 3.0 (inte en bild eller SWF-fil som är skriven i ActionScript 1.0 eller 2.0). Vid inläsningen av filen kan du ange att filen ska placeras i en specifik programdomän, i stället för att med standardinställningen placeras i en ny programdomän som är underordnad den inläsande SWF-filens programdomän. Observera, att programdomäner är underenheter till säkerhetsdomäner och därför kan du ange en destinationsprogramdomän endast om SWF-filen du läser in kommer från den egna säkerhetsdomänen, beroende på antingen att den kommer den från den egna servern eller att importen har slutförts till den egna säkerhetsdomänen med egenskapen `securityDomain`. Om du anger en programdomän, men den inlästa SWF-filen ingår i en annan säkerhetsdomän, ignoreras angiven domän i `applicationDomain`. Mer information finns i ”Arbeta med programdomäner” på sidan 148.

Mer information finns i ”Ange innehåll för inläsning” på sidan 200.

En viktig egenskap för Loader-objektet är egenskapen `contentLoaderInfo`, vilket är ett LoaderInfo-objekt. Till skillnad från de flesta andra objekt delas LoaderInfo-objektet mellan inläsande SWF-fil och det inlästa innehållet. Objektet är dessutom alltid tillgängligt för båda parter. Om det inlästa innehållet är en SWF-fil har den åtkomst till objektet LoaderInfo via egenskapen `DisplayObject.loaderInfo`. LoaderInfo-objekt innehåller uppgifter om inläsningsförloppet, URL-adresserna för in- och utläsare, tillförlitligheten mellan in- och utläsare samt annan information. Mer information finns i ”Övervaka inläsningsförloppet” på sidan 199.

Inläsning av ljud och video

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Allt innehåll, förutom det som finns i sandlådan lokal-med-filsystem, tillåts att läsa in ljud och video från nätverksplatser med metoderna `Sound.load()`, `NetConnection.connect()` och `NetStream.play()`.

Endast innehåll i sandlådan lokal-med-filsystem och i AIR-programsandlådor kan läsa in media från det lokala filsystemet. Endast innehåll i sandlådan lokal-med-filsystem och i AIR-programsandlådan eller sandlådan lokal-tillförlitlig har åtkomst till data i dessa inlästa filer.

Det finns andra begränsningar för åtkomst av data från inlästa medier. Mer information finns i ”Åtkomst av inlästa medier som data” på sidan 1070.

Inläsning av SWF-filer och bilder via taggen <img> i ett textfält

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan läsa in SWF-filer och bitmappar till ett textfält genom att använda taggen `<img>` på följande sätt:

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Åtkomst till det inlästa innehållet ges via metoden `getImageReference()` i TextField-instansen, enligt följande:

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

Observera däremot att SWF-filer och bilder som är inlästa på detta sätt placeras i sandlådan som motsvarar deras ursprung.

När du läser in en bildfil med hjälp av taggen `<img>` i ett textfält ges åtkomsten till data i bilden via en URL-principfil. Du kan söka efter en principfil genom att lägga till attributet `checkPolicyFile` i taggen `<img>`, enligt följande:

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

När du läser in en SWF-fil med hjälp av taggen `<img>` i ett textfält kan du tillåta åtkomst till SWF-filens data via ett anrop till metoden `Security.allowDomain()`.

När du använder taggen `<img>` i ett textfält vid inläsning av en extern fil (till skillnad mot att använda klassen Bitmap inbäddad i SWF) skapas automatiskt ett Loader-objekt som underordnat till TextField-objektet. Den externa filen läses in i Loader-objektet precis som om du hade använt ett Loader-objekt i ActionScript. I det här fallet returnerar metoden `getImageReference()` Loader-objektet som skapades automatiskt. Ingen säkerhetskontroll behövs vid åtkomst av detta Loader-objekt eftersom det ligger i samma säkerhetssandlåda som anropande kod.

Däremot gäller säkerhetsreglerna när du refererar till egenskapen `content` i Loader-objektet vid åtkomst till inlästa medier. Om innehållet är en bild måste du implementera en URL-principfilen och om innehållet är en SWF-fil måste metoden `allowDomain()` anropas av koden i SWF-filen.

Adobe AIR

I programsandlådan ignoreras <img>-taggar i ett textfält för att undvika nätfiskeattacker. Kod som körs i programsandlådan har inte heller behörighet att anropa metoden `allowDomain()`.

Innehåll som levereras via RTMP-servrar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash Media Server används RTMP (Real-Time Media Protocol) för att återge data, ljud och video. Du kan läsa in detta medieinnehåll genom att använda metoden `connect()` för klassen NetConnection och skicka RTMP URL som parameter. Flash Media Server kan begränsa anslutningar och förhindra att innehåll laddas ned, baserat på den begärande filens domän. Mer information finns i dokumentationen till Flash Media Server på www.adobe.com/go/learn_fms_docs_se.

Om du vill använda metoderna `BitmapData.draw()`, `BitmapData.drawWithQuality()` och `SoundMixer.computeSpectrum()` för att extrahera körningsgrafik och ljuddata från RTMP-strömmar måste tillåta tillgång till servern. Använd egenskaperna `Client.videoSampleAccess` och `Client.audioSampleAccess` för ActionScript på serversidan om du vill tillåta åtkomst till särskilda kataloger på Flash Media Server. Mer information finns i Språkreferens för ActionScript på server. (Metoden `drawWithQuality` är tillgänglig i Flash Player 11.3 och senare; AIR 3.3 och senare.)

Korsskriptning

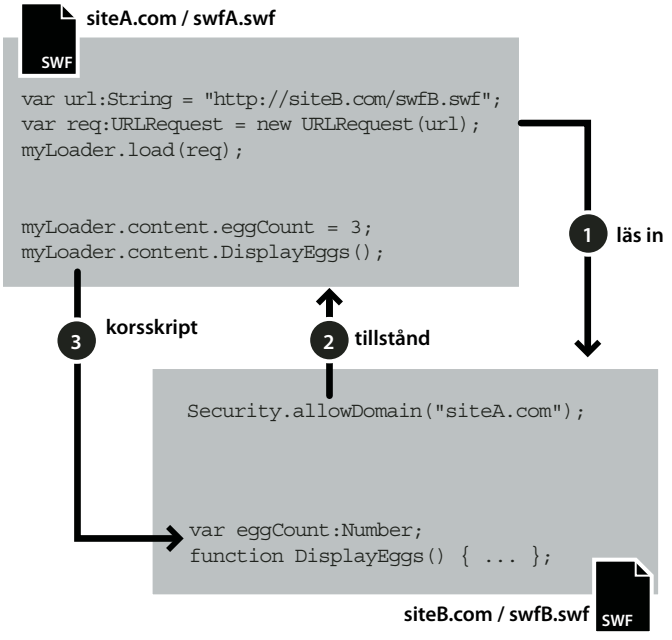
Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om två SWF-filer som är skrivna med ActionScript 3.0, eller två HTML-filer som körs i AIR opereras från samma domän, där URL:en för den ena SWF-filen till exempel är http://www.example.com/swfA.swf och för den andra http://www.example.com/swfB.swf, kan kod som definieras i den ena filen undersöka och ändra variabler, objekt, egenskaper, metoder o.s.v. i den andra och vice versa. Detta kallas *korsskriptning*.

De två filerna kan också opereras från olika domäner, till exempel http://siteA.com/swfA.swf och http://siteB.com/swfB.swf. I sådana fall är standard att Flash Player och AIR inte tillåter att swfA.swf skriptar swfB.swf, eller att swfB.swf skriptar swfA.swf. En SWF-fil ger SWF-filer från andra domäner behörighet genom att anropa `Security.allowDomain()`. Genom att anropa `Security.allowDomain("siteA.com")` ger siteB.swf SWF-filer från siteA.swf behörighet att skripta den.

Korsskriptning används inte mellan AVM1 SWF- och AVM2 SWF-filer. En AVM1 SWF-fil har skapats med ActionScript 1.0 eller ActionScript 2.0. (AVM1 och AVM2 betyder ActionScript Virtual Machine.) Däremot kan du använda klassen LocalConnection för att skicka data mellan AVM1 och AVM2.

I alla korsdomänssituationer är det viktigt att veta vilken sida som är vilken. I beskrivningarna kommer sidan som utför korsskriptningen att kallas *den anslutande parten* (vanligtvis den anslutande SWF-filen) och den andra sidan kallas *målparten* (vanligtvis SWF-målfilen). När siteA.swf skriptar siteB.swf är siteA.swf den anslutande parten och siteB.swf målparten, som illustreras här:



Korsdomänbehörigheter som upprättas med metoden `Security.allowDomain()` är asymmetriska. I föregående exempel kan siteA.swf skripta siteB.swf, men siteB.swf kan inte skripta siteA.swf eftersom siteA.swf inte har anropat metoden `Security.allowDomain()` för att ge SWF-filer på siteB.com behörighet att skripta den. Du kan ange symmetrisk behörighet genom att låta båda SWF-filerna anropa metoden `Security.allowDomain()`.

Flash Player skyddar SWF-filer från korsdomänsskriptning från andra SWF-filer och från korsdomänsskriptning från HTML-filer. HTML-till-SWF-skriptning kan utföras med återkopplingar (callback) som upprättats via metoden `ExternalInterface.addCallback()`. När skriptning från HTML till SWF går över domäner, måste SWF-målfilen anropa metoden `Security.allowDomain()`, precis som när den anslutande parten är en SWF-fil, annars misslyckas åtgärden. Mer information finns i ”Författarinställningar (för utvecklare)” på sidan 1059.

I Flash Player finns dessutom säkerhetskontroller för SWF-till-HTML-skriptning. Mer information finns i ”Kontrollera utgående URL-åtkomst” på sidan 1077.

Scen, säkerhet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Vissa egenskaper och metoder i objektet Stage kan användas med Sprite- eller MovieClip-objekt i visningslistan.

Objektet Stage har däremot en ägare: den först inlästa SWF-filen. Följande egenskaper och metoder i objektet Stage är som standard endast tillgängliga för SWF-filer i samma säkerhetssandlåda, som är objektsägare:

Egenskaper	Metoder
`align`	`addChild()`
displayState	`addChildAt()`
frameRate	`addEventListener()`
`height`	`dispatchEvent()`
`mouseChildren`	`hasEventListener()`
`numChildren`	`setChildIndex()`
`quality`	`willTrigger()`
`scaleMode`	
`showDefaultContextMenu`	
`stageFocusRect`	
`stageHeight`	
`stageWidth`	
`tabChildren`	
`textSnapshot`	
`width`	

För att en SWF-fil i en annan sandlåda än gällande objektsägares sandlåda ska få tillgång till dessa egenskaper och metoder, måste scenägarens SWF-fil anropa metoden `Security.allowDomain()` och ge behörighet till domänen för den externa sandlådan. Mer information finns i ”Författarinställningar (för utvecklare)” på sidan 1059.

Egenskapen `frameRate` är ett undantag – samtliga SWF-filer kan läsa egenskapen `frameRate`. Däremot kan endast de filer som finns i objektsägarens säkerhetssandlåda (eller de som har behörighet via anrop av metoden `Security.allowDomain()`) ändra egenskapen.

Det finns även begränsningar för metoderna `removeChildAt()` och `swapChildrenAt()` i Stage-objektet, men de skiljer sig från andra begränsningar. För att anropa dessa metoder måste koden finnas i samma domän som ägaren av berörda underordnade objekt, alternativt kan underordnade objekt anropa metoden `Security.allowDomain()`, i stället för att de måste ligga i samma domän som objektsägaren.

Gå igenom visningslistan

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Möjligheten att en SWF-fil har åtkomst till visningsobjekt som är inlästa från andra sandlådor är begränsad. För att en SWF-fil ska få åtkomst till ett visningsobjekt som skapats av en annan SWF-fil i en annan sandlåda, måste SWF-filen anropa metoden `Security.allowDomain()` för att ge behörighet från domänen. Mer information finns i ”Författarinställningar (för utvecklare)” på sidan 1059.

För att få åtkomst till Bitmap-objektet som lästes in med Loader-objektet måste det finnas en URL-principfil på bildfilens ursprungliga server. Denna korsdomänprincip måste ge behörighet till domänen med SWF-filen som försöker få åtkomst till Bitmap-objektet (se”Webbplatsinställningar (principfiler)” på sidan 1056).

Objektet LoaderInfo som har en motsvarande inläst fil (och ett Loader-objekt) innehåller följande tre egenskaper som definierar relationen mellan det inlästa objektet och Loader-objektet: `childAllowsParent`, `parentAllowsChild` och `sameDomain`.

Säkerhet med händelser

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Händelser som berör visningslistan har begränsningar för åtkomst, baserat på sandlådan varifrån visningsobjektet skickar händelsen. En händelse i visningslistan har bubblings- och hämtningsfaser (beskrivs i "Hantera händelser" på sidan 125). Under bubblings- och hämtningsfaserna flyttas en händelse från källvisningsobjektet via överordnade visningsobjekt i visningslistan. Om ett överordnat objekt ligger i en annan säkerhetssandlåda än källvisningsobjektet stannar bubblings- och hämtningsfasen nedanför det överordnade objektet, såvida det inte finns inbördes tillförlitlighet hos både ägaren av överordnat objekt och ägaren av källobjekt. En sådan inbördes tillförlitlighet skapas genom följande:

1. SWF-filen som äger det överordnade objektet måste anropa metoden `Security.allowDomain()` för att göra domänen med SWF-filen som äger källobjektet tillförlitlig.
2. SWF-filen som äger källobjektet måste anropa metoden `Security.allowDomain()` för att göra domänen med SWF-filen som äger det överordnade objektet tillförlitligt.

Objektet LoaderInfo som har en motsvarande inläst fil (och ett Loader-objekt) innehåller följande två egenskaper som definierar relationen mellan det inlästa objektet och Loader-objektet: `childAllowsParent` och `parentAllowsChild`.

För händelser som skickas från andra objekt än visningsobjekt finns det inga säkerhetskontroller eller säkerhetsrelaterade konsekvenser.

Åtkomst av inlästa medier som data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om du vill komma åt inläsningsdata använder du metoderna `BitmapData.draw()`, `BitmapData.drawWithQuality()` och `SoundMixer.computeSpectrum()`. Som standard ges inte åtkomst till pixeldata eller ljuddata från grafik- eller ljudobjekt som återges eller används av media som har lästs in i en annan sandlåda. Men du kan använda följande metoder om du vill ge åtkomst till sådana data över sandlådegränserna:

- Anropa metoden `Security.allowDomain()` under återgivningen av det innehåll eller uppspelningen av de data som du vill ge åtkomst till för att ge åtkomst till innehåll i andra domäner.
- För inläst bild, ljud eller video lägger du till en URL-principfil på servern med den inlästa filen. Den här principfilen måste ge tillgång till domänen för den SWF-fil som försöker anropa metoderna `BitmapData.draw()`, `BitmapData.drawWithQuality()` eller `SoundMixer.computeSpectrum()` för att extrahera data från filen. Metoden `drawWithQuality` är tillgänglig i Flash Player 11.3 och senare; AIR 3.3 och senare.

I följande avsnitt beskrivs hur du får åtkomst till bitmapps-, ljud- och videodata.

Åtkomst till bitmappsdata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med metoderna `draw()` och `drawWithQuality()` (Flash Player 11.3; AIR 3.3) för ett BitmapData-objekt kan du rita de pixlar som för tillfället visas för ett visningsobjekt till BitmapData-objektet. Detta kan innehålla pixlarna från ett MovieClip-objekt, ett Bitmap-objekt eller något annat visningsobjekt. Följande villkor måste uppfyllas för de här metoderna för att pixlar ska kunna ritas till BitmapData-objektet:

- Om det gäller ett annat källobjekt än en inläst bitmapp måste källobjektet och (om det är ett Sprite- eller MovieClip-objekt) alla underordnade objekt komma från samma domän som det objekt som anropar metoden draw eller så måste de finnas i en SWF-fil som är tillgänglig för anroparen via metoden `Security.allowDomain()`.
- Om det gäller ett källobjekt av typen inläst bitmapp måste källobjektet komma från samma domän som det objekt som anropar metoden draw eller så måste källservern ha en URL-principfil som ger den anropande domänen tillgång till servern.

Om villkoren inte uppfylls genereras ett SecurityError-undantag.

Om du läser in bilden med metoden `load()` i klassen Loader kan du ange parametern `context`, som är ett LoaderContext-objekt. Om du ställer in egenskapen `checkPolicyFile` för LoaderContext-objektet på `true` söker Flash Player efter en URL-principfil på servern som bilden läses in på. Om det finns en principfil och om filen godkänner domänen för den inläsande SWF-filen, får filen åtkomst till data i Bitmap-objektet, annars får den inte åtkomst.

Du kan även ange egenskapen `checkPolicyFile` i en bild som lästs in via taggen `<img>` i ett textfält. Mer information finns i ”Inläsning av SWF-filer och bilder via taggen <img> i ett textfält” på sidan 1066.

Åtkomst av ljuddata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Följande ljudrelaterade API:er i ActionScript 3.0 har säkerhetsbegränsningar:

- Metoden `SoundMixer.computeSpectrum()` – Tillåts alltid för kod som körs i samma säkerhetssandlåda som ljudfilen. För kod som körs i andra sandlådor används säkerhetskontroller.
- Metoden `SoundMixer.stopAll()` – Tillåts alltid för kod som körs i samma säkerhetssandlåda som ljudfilen. För filer i andra sandlådor finns det säkerhetskontroller.
- Egenskapen `id3` för klassen Sound – tillåts alltid för SWF-filer som finns i samma säkerhetssandlåda som ljudfilen. För kod som körs i andra sandlådor används säkerhetskontroller.

Varje ljud har två typer av associerade sandlådor; en för innehållet och en för ägaren.

- Den ursprungliga domänen för ljudet utgör innehållssandlådan och denna avgör om data kan extraheras från ljudet via egenskapen `id3` och metoden `SoundMixer.computeSpectrum()`.
- Objektet som startade ljuduppspelningen utgör ägarsandlådan och detta avgör om ljudet kan stoppas via metoden `SoundMixer.stopAll()`.

Om du läser in ljudet med metoden `load()` i klassen Sound kan du ange parametern `context`, som är ett SoundLoaderContext-objekt. Om du anger egenskapen `checkPolicyFile` för SoundLoaderContext-objektet till `true` söker körningen efter en URL-principfil på den server från vilken ljudet lästes in. Om det finns en principfil som godkänner domänen för den kod som ska läsas in får koden åtkomst till egenskapen `id` för Sound-objektet, annars får den inte åtkomst. Genom att dessutom ange egenskapen `checkPolicyFile` kan metoden `SoundMixer.computeSpectrum()` användas för inlästa ljud.

Du kan använda metoden `SoundMixer.areSoundsInaccessible()` för att ta reda på om ett anrop till metoden `SoundMixer.stopAll()` inte kommer att stoppa alla ljud, på grund av att anroparen saknar åtkomst till sandlådan för en eller flera ljudägare.

Genom anrop av metoden `SoundMixer.stopAll()` stoppas dessa ljud vars ägarsandlåda är densamma som för den som anropar `stopAll()`. Den stoppar även de ljud vars uppspelning startades med SWF-filer som har anropat metoden `Security.allowDomain()` för att tillåta åtkomst av domänen med SWF-filen som anropar metoden `stopAll()`. Några andra ljud stoppas inte och förekomsten av dessa ljud kan upptäckas genom anrop av metoden `SoundMixer.areSoundsInaccessible()`.

För anrop av metoden `computeSpectrum()` krävs att alla ljud som spelas upp antingen är från samma sandlåda som anropande objekt av metoden eller från en källa som har tilldelats behörighet till anroparens sandlåda, i annat fall genereras ett SecurityError-undantag. För ljud som lästs in från inbäddade ljud i ett bibliotek i en SWF-fil tilldelas behörighet genom ett anrop till metoden `Security.allowDomain()` i den inlästa SWF-filen. För ljud som lästs in från andra källor än SWF-filer (MP3-filer eller videofiler) är det en URL-principfil på källservern som beviljar tillgång till data i inlästa medier.

Mer information finns i ”Författarinställningar (för utvecklare)” på sidan 1059 och ”Webbplatsinställningar (principfiler)” på sidan 1056.

För åtkomst till ljuddata från RTMP-strömmar måste du tillåta åtkomst på servern. Använd egenskapen `Client.audioSampleAccess` för ActionScript på serversidan om du vill tillåta åtkomst till särskilda kataloger på Flash Media Server. Mer information finns i Språkreferens för ActionScript på server.

Komma åt videodata

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan använda metoden `BitmapData.draw()` eller `BitmapData.drawWithQuality()` för att hämta pixeldata för den aktuella bildrutan i en video. (Metoden `drawWithQuality` är tillgänglig i Flash Player 11.3 och senare; AIR 3.3 och senare.)

Det finns två typer av video:

- Direktuppspelad video via RTMP från Flash Media Server
- Progressiv video, som läses in från en FLV- eller F4V-fil

Om du vill använda draw-metoderna för att extrahera körningsgrafik från RTMP-strömmar måste du tillåta tillgång till servern. Använd egenskapen `Client.videoSampleAccess` för ActionScript på serversidan om du vill tillåta åtkomst till särskilda kataloger på Flash Media Server. Mer information finns i Språkreferens för ActionScript på server.

När du anropar en draw-metod med progressiv video som `source`-parameter måste anroparen antingen finnas i samma sandlåda som FLV-filen eller så måste FLV-filens server ha en principfil som ger den anropande SWF-filens domän tillgång till servern. Du kan begära att principfilen ska laddas ned genom att ange egenskapen `checkPolicyFile` för NetStream-objektet med `true`.

Inläsning av data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Flash Player- och AIR-innehåll kan utbyta data med en server. Inläsning av data är en annan typ av åtgärd än inläsning av medier, eftersom den inlästa informationen visas som programobjekt i stället för att visas som media. Vanligtvis kan innehåll läsa in data från samma domän som innehållet kom från. Däremot krävs det vanligtvis principfiler om innehållet ska kunna läsa in data från andra domäner (se ”Webbplatsinställningar (principfiler)” på sidan 1056).

Obs! *Innehåll som körs i AIR-programmets sandlåda serveras aldrig från en fjärrdomän (om inte utvecklaren avsiktligen importerar fjärrinnehåll till programmets sandlåda), och det kan därför inte delta i de typer av attacker som principfiler skyddar mot. AIR-innehåll i programsandlådan är inte begränsat från att läsa in data genom principfiler. Med AIR-innehåll i andra sandlådor har de begränsningar som beskrivs här.*

Använda URLLoader och URLStream

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du kan läsa in data som till exempel en XML-fil eller textfil. Metoderna `load()` för klasserna URLLoader och URLStream styrs av behörigheter i URL-principfiler.

Om du använder metoden `load()` för att läsa in innehåll från en annan domän än den med koden som anropar metoden, söker körningen efter en URL-principfil på servern med de inlästa resurserna. Om det finns en principfil som ger åtkomst till domänen för det innehåll som ska läsas in kan du läsa in dessa data.

Ansluta till socketar

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Körningen söker som standard efter en socketprincipfil som hanteras från port 843. Liksom med URL-principfiler kallas den här filen för *huvudprincipfil*.

När principfiler först började användas i Flash Player 6 fanns det inget stöd för socketprincipfiler. Anslutningar till socketservrar fick behörighet genom en principfil i standardplatsen på en HTTP-server på port 80 för samma värddator som socketservern. Flash Player 9 stöder fortfarande den här funktionen men Flash Player 10 gör det inte. I Flash Player 10 kan bara socketprincipfiler auktorisera socketanslutningar.

Precis som URL-principfiler stöder socketprincipfiler en metaprincipprogramsats, som anger vilka portar som kan hantera principfiler. I stället för att vara en ”master-only” är standardmetaprincipen för socketprincipen ”all”. Såvida inte huvudprincipfilen anger en mer restriktiv inställning förutsätter Flash Player att valfri socket på värden kan hantera en socketprincipfil.

Åtkomsten till socket- och XML-socketanslutningar inaktiveras som standard, även om den socket som du ansluter till finns i samma domän som SWF-filen. Du kan tillåta åtkomst på socketnivå genom att hantera en socketprincipfil från någon av följande platser:

- Port 843 (huvudprincipfilens placering)
- Samma port som huvudsocketanslutningen
- En annan port än huvudsocketanslutningen

Flash Player söker som standard efter en socketprincipfil på port 843 och på samma port som huvudsocketanslutningen. Om du vill hantera en socketprincipfil från en annan port måste SWF-filen anropa `Security.loadPolicyFile()`.

En socketprincipfil har samma syntax som en URL-principfil, förutom att den också måste ange vilka portar som ska vara öppna. När en socketprincipfil hanteras från en port med ett lägre nummer än 1024 kan den ge åtkomst till alla portar. När en principfil kommer från port 1024 eller högre kan den bara ge åtkomst till port 1024 och högre. De tillåtna portarna anges i ett `to-ports`-attribut i taggen `<allow-access-from>`. Enstaka portnummer, portintervall och jokertecken får användas.

Följande är ett exempel på en socketprincipfil:

```
<?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>
```

Om du vill hämta en socketprincipfil från port 843 eller från samma port som huvudsocketanslutningen anropar du metoden `Socket.connect()` eller `XMLSocket.connect()`. Flash Player söker först efter en huvudprincipfil på port 843. Om det finns någon där, kontrollerar programmet om filen innehåller en metaprincipprogramsats som förbjuder socketprincipfiler på målporten. Om åtkomsten inte förbjuds söker Flash Player först efter en lämplig `allow-access-from`-sats i huvudprincipfilen. Om det inte finns någon, söker programmet efter en socketprincipfil på samma port som huvudsocketanslutningen.

Om du vill hämta en socketprincipfil på en annan plats anropar du först metoden `Security.loadPolicyFile()` med den speciella `"xmlsocket"`-syntaxen, så här:

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Anropa metoden `Security.loadPolicyFile()` innan anrop av metoden `Socket.connect()` eller `XMLSocket.connect()` görs. Därefter väntar Flash Player tills principfilens begäran är slutförd innan programmet avgör om huvudanslutningen tillåts. Om huvudprincipfilen anger att målplatsen kan hantera principfiler, har anropet till `loadPolicyFile()` ingen effekt, även om det finns en principfil på den platsen.

Om du implementerar en socketserver och du behöver använda en socketprincip, måste du bestämma om du ska använda principfilen via samma port som tillåter huvudanslutningar eller en annan port. I båda fallen måste servern vänta på den första överföringen från klienten innan ett svar skickas.

Om Flash Player begär en principfil skickas alltid följande sträng så snart anslutningen är upprättad:

```
<policy-file-request/>
```

När servern tar emot den här strängen kan principfilen skickas. Förfrågningen från Flash Player avslutas alltid med en null-byte, och svaret från servern måste också avslutas med en null-byte.

Du kan inte återanvända samma anslutning för både en principfilsbegäran och en huvudanslutning. Stäng anslutningen när principfilen har skickats. Om du inte gör det avslutas principfilsanslutningen av Flash Player innan huvudanslutningen upprättas.

Skydda data

Flash Player 9 och senare, Adobe AIR 1.0 och senare

För att skydda data när de skickas över Internet kan du använda TLS (Transport Layer Security) eller SSL (Secure Sockets Layer) på den server varifrån informationen kommer. Sedan kan du ansluta till servern med HTTPS-protokollet.

I program som skapats för AIR 2 eller senare versioner kan du även skydda TCP-socketkommunikation. Med klassen SecureSocket kan du initiera en socketanslutning till en socketserver som använder TLS version 1 eller SSL version 4.

Datasändning

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Data skickas när koden skickar dessa till en server eller en resurs. Det är alltid tillåtet att skicka data för innehåll från en nätverksdomän. En lokal SWF-fil kan endast skicka data till nätverksadresser om den ligger i sandlådan lokal-tillförlitlig eller lokal-med-nätverk, eller i AIR-programsandlådan. Mer information finns i ”Lokala sandlådor” på sidan 1049.

Du kan använda funktionen `flash.net.sendToURL()` för att skicka data till en URL-adress. Det finns andra metoder som också skickar begäran till URL-adresser. Det är inläsningsmetoder, t.ex. `Loader.load()` och `Sound.load()`, och datainläsningsmetoder, t.ex. `URLLoader.load()` och `URLStream.load()`.

Överföra och ladda ned filer

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Metoden `FileReference.upload()` startar överföringen av en fil som angetts av en användare till en extern server. Innan du anropar metoden `FileReference.upload()` måste du anropa metoden `FileReference.browse()` eller `FileReferenceList.browse()`.

Koden som initierar metoden `FileReference.browse()` eller `FileReferenceList.browse()` kan bara anropas som svar på en mushändelse eller en tangentbordshändelse. Om koden anropas i andra situationer genereras ett undantag i Flash Player 10 och senare. Men en användarinitierad händelse krävs inte för att anropa dessa metoder från AIR-programsandlådan.

Anrop av metoden `FileReference.download()` öppnar en dialogruta där användaren kan ladda ned en fil från en extern server.

Obs! *Om servern kräver användarverifiering kan bara SWF-filer som körs i en webbläsare (d.v.s. med webbläsarplugin eller ActiveX-kontroll) öppna en dialogruta där användaren kan ange användarnamn och lösenord för verifiering, och bara för nedladdning. Inga överföringar kan göras till en server som kräver användarverifiering.*

Överföring är inte tillåtet om den anropande SWF-filen körs i sandlådan lokal-med-filsystem.

Som standard kan en SWF-fil inte initiera en överföring till, eller nedladdning från, någon annan server än den egna. En SWF-fil kan överföra till, eller ladda ned från, en annan server om den har en principfil som ger behörighet till SWF-filens domän.

Inläsning av inbäddat innehåll från SWF-filer som har importerats till en säkerhetsdomän

Flash Player 9 och senare, Adobe AIR 1.0 och senare

När du läser in en SWF-fil kan du ange parametern `context` för metoden `load()` i Loader-objektet som används vid inläsningen av filen. Denna parameter tar emot objektet LoaderContext. Om du anger egenskapen `securityDomain` för LoaderContext-objektet med `Security.currentDomain` söker Flash Player efter en URL-principfil på servern med inläst SWF-fil. Om det finns en principfil som ger åtkomst till domänen med inläsande SWF-fil kan du läsa in SWF-filen som importerat medium. På så vis kan den inläsande filen få åtkomst till objekt i SWF-filens bibliotek.

Ett alternativt sätt för en SWF-fil att få åtkomst till klasser i inlästa SWF-filer från en annan säkerhetssandlåda är att den inlästa SWF-filen anropar metoden `Security.allowDomain()` för att ge åtkomst till domänen med anropande SWF-fil. Du kan lägga till anropet till metoden `Security.allowDomain()` till konstruktormetoden för den inlästa SWF-filens huvudklass och därefter se till att inläsande SWF-fil lägger till en händelseavlyssnare för händelsen `init` som skickas av egenskapen `contentLoaderInfo` för Loader-objektet. När denna händelse skickas har den inlästa SWF-filen anropat metoden `Security.allowDomain()` i konstruktormetoden och klasser i den inlästa SWF-filen är tillgängliga för inläsande SWF-fil. Den inläsande SWF-filen kan hämta klasser från den inlästa SWF-filen genom att anropa `Loader.contentLoaderInfo.applicationDomain.getDefinition()` eller Loader.contentLoaderInfo.applicationDomain.getQualifiedDefinitionNames() (Flash Player 11.3 och senare; AIR 3.3 och senare).

Arbeta med äldre innehåll

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash Player 6 baserades domänen som används för vissa inställningar i Flash Player på den avslutande delen i SWF-filens domän. Dessa inställningar avser behörighet för kamera och mikrofon, lagringskvoter och lagring av beständiga delade objekt.

Om en SWF-fils domän består av mer än två segment, t.ex. www.example.com, tas det första segmentet bort (www) och den återstående delen används. I Flash Player 6 används därför example.com som domän för både www.example.com och store.example.com med de här inställningarna. På samma sätt används example.co.uk som domän för både www.example.co.uk and store.example.co.uk med de här inställningarna. Detta kan ge problem med att SWF-filer från orelaterade domäner, t.ex. example1.co.uk och example2.co.uk, har åtkomst till samma delade objekt.

I Flash Player 7 och senare ges spelarinställningarna standardvärden enligt SWF-filens exakta domän. En SWF-fil från www.example.com skulle t.ex. använda spelarinställningar för www.example.com. En SWF-fil från store.example.com skulle använda separata spelarinställningar för store.example.com.

I en SWF-fil som är skriven med ActionScript 3.0, där `Security.exactSettings` anges med `true` (standard), använder Flash Player exakta domäner för spelarinställningar. Om `false` anges använder Flash Player domäninställningarna som används i Flash Player 6. Om du ändrar `exactSettings` från standardvärdet måste du göra detta innan några händelser inträffar som gör att Flash Player väljer spelarinställningar, t.ex. vid användning av kamera eller mikrofon, eller hämtning av ett beständigt delat objekt.

Om du har publicerat en SWF-fil version 6 och skapat bestående delade objekt därifrån, och vill hämta dessa objekt från en SWF-fil som används med ActionScript 3.0, måste du ange `Security.exactSettings` med `false` innan du anropar `SharedObject.getLocal()`.

Inställning av LocalConnection-behörighet

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Med klassen LocalConnection kan du skicka meddelanden mellan olika Flash Player-program eller AIR-program. LocalConnection-objekt kan endast kommunicera med Flash Player- eller AIR-innehåll som körs på samma klientdator, men de kan köras i olika program – till exempel en SWF-fil som körs i en webbläsare eller en SWF-fil som körs i en projektor – och ett AIR-program kan kommunicera med alla med klassen LocalConnection.

För varje LocalConnection-kommunikation finns det en avsändare och en avlyssnare. Som standard tillåter Flash Player LocalConnection-kommunikation mellan kod som körs inom samma domän. För kod som körs i olika sandlådor måste avlyssnaren tillåta sändarbehörighet via metoden `LocalConnection.allowDomain()`. Strängen som du skickar som argument till metoden `LocalConnection.allowDomain()` kan innehålla något av följande: exakta domännamn, IP-adresser och jokertecknet `*`.

Metoden `allowDomain()` har ändrats från formatet som användes i ActionScript 1.0 och 2.0. I dessa tidigare versioner var `allowDomain()` en återkopplingsmetod som du själv implementerade. I ActionScript 3.0 är `allowDomain()` en inbyggd metod i klassen LocalConnection som du anropar. Förutom denna skillnad fungerar `allowDomain()` nästan likadant som `Security.allowDomain()`.

En SWF-fil kan använda egenskapen `domain` för klassen LocalConnection för att avgöra domän.

Kontrollera utgående URL-åtkomst

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Du uppnår utgående skriptning och URL-åtkomst (med hjälp av HTTP URL:er, mailto: o.s.v.) genom att använda följande API:er:

- Funktionen `flash.system.fscommand()`
- Metoden `ExternalInterface.call()`
- Funktionen `flash.net.navigateToURL()`

För innehåll som läses in från det lokala filsystemet kan endast dessa metoder anropas om koden och den webbsidan som fungerar som behållare (om det finns någon) finns i säkerhetssandlådan lokal-tillförlitlig eller säkerhetssandlådan för AIR-programmet. Anrop till dessa metoder misslyckas om innehållet finns i sandlådorna lokal-med-nätverk eller lokal-med-filsystem.

För innehåll som inte läses in lokalt kan alla dessa API:er kommunicera med webbsidan där de är inbäddade, beroende på värdet på AllowScriptAccess-parametern som beskrivs nedan. Funktionen `flash.net.navigateToURL()` kan också kommunicera med alla öppna webbläsarfönster eller -ramar, inte bara sidan där SWF-filen är inbäddad. Mer information om den här funktionen finns i ”Använda funktionen navigateToURL()” på sidan 1078.

Parametern `AllowScriptAccess` i HTML-koden som läser in en SWF-fil kontrollerar möjligheten att utföra utgående URL-åtkomst inifrån SWF-filen. Ange den här parametern i taggen PARAM eller EMBED. Om du inte anger något värde för `AllowScriptAccess` kan SWF-filen och HTML-sidan bara kommunicera om båda kommer från samma domän.

Parametern `AllowScriptAccess` kan anges med någon av dessa tre värden: `"always"`, `"sameDomain"` eller `"never"`.

- När `AllowScriptAccess` är inställd på `"always"` kan SWF-filen kommunicera med HTML-sidan som den är inbäddad i även när SWF-filen kommer från en annan domän än HTML-sidan.

- När `AllowScriptAccess` är inställd på `"sameDomain"` kan SWF-filen bara kommunicera med HTML-sidan som den är inbäddad i när SWF-filen kommer från samma domän som HTML-sidan. Det här värdet är standardvärdet för `AllowScriptAccess`. Använd den här inställningen, eller ange inte något värde för `AllowScriptAccess`, om du vill förhindra att en SWF-fil som finns på en domän får tillgång till ett skript på en HTML-sida som kommer från en annan domän.
- Om `AllowScriptAccess` ställs in på `"never"` kan inte SWF-filen kommunicera med någon HTML-sida. Användningen av det här värdet har tagits bort från och med Adobe Flash CS4 Professional. Det rekommenderas inte och bör inte vara nödvändigt om du inte hanterar ej betrodda SWF-filer från din egen domän. Om du måste skicka SWF-filer som du inte litar på rekommenderar Adobe att du skapar en separat underdomän och placerar allt innehåll som du inte litar på där.

Nedan finns ett exempel på hur du ställer in taggen `AllowScriptAccess` på en HTML-sida så att utgående URL-åtkomst tillåts till en annan domän:

```
<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='always'/>
<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>
```

Använda funktionen navigateToURL()

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Förutom säkerhetsinställningen som anges med parametern `allowScriptAccess` och som beskrivs ovan, har funktionen `navigateToURL()` en till parameter – `target`. Parametern `target` kan användas för att ange namnet på ett HTML-fönster eller en bildruta att skicka URL-förfrågningen till. Flera säkerhetsbegränsningar gäller för sådana förfrågningar, och begränsningarna varierar beroende på om `navigateToURL()` används som en skriptsats eller annan typ av programsats.

För skriptsatser, till exempel `navigateToURL("javascript: alert('Hälsningar från Flash Player.')")`, gäller följande regler.

- Om SWF-filen är en lokalt betrodd fil, lyckas förfrågningen.
- Om målet är HTML-sidan där SWF-filen bäddas in, gäller de `allowScriptAccess`-regler som beskrivs ovan.
- Om målet innehåller innehåll som läses in från samma domän som SWF-filen, lyckas förfrågningen.
- Om målet innehåller innehåll som läses in från en annan domän än SWF-filen, och inget av de tidigare villkoren uppfylls, misslyckas förfrågningen.

För andra typer av programsatser än skriptsatser (till exempel HTTP, HTTPS och `mailto:`) misslyckas förfrågningen om alla av följande villkor gäller:

- Målet är något av specialnyckelorden `"_top"` eller `"_parent"`, och
- SWF-filen finns på en webbsida i en annan domän, och
- SWF-filen bäddas in med ett värde på `allowScriptAccess` som inte är `"always"`.

Mer information

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Mer information om utgående URL-åtkomst finns följande avsnitt i *Referenshandbok för ActionScript 3.0 i Adobe Flash-plattformen*:

- Funktionen flash.system.fscommand()
- Metoden call() för klassen ExternalInterface
- Funktionen `flash.net.navigateToURL()`

Delade objekt

Flash Player 9 och senare, Adobe AIR 1.0 och senare

I Flash Player kan du använda *delade objekt*, som är ActionScript-objekt som finns utanför en SWF-fil, antingen lokalt på användarens filsystem eller externt på en RTMP-server. Delade objekt, liksom andra medier i Flash Player, delas upp i säkerhetssandlådor. Däremot skiljer sig sandlådemodellen för delade objekt något, eftersom delade objekt är resurser som aldrig kan göras tillgängliga över domängränser. I stället hämtas alltid delade objekt från ett lager med delade objekt som är specifikt för domänen för respektive SWF-fil som anropar metoder för klassen SharedObject. Vanligtvis är ett lager med delade objekt ännu mer specifikt än en SWF-fils domän: alla SWF-filer använder som standard ett lager med delade objekt som är specifikt för hela den ursprungliga URL-adressen. Mer information om delade objekt finns i ”Delade objekt” på sidan 703.

En SWF-fil kan använda parametern `localPath` för metoden `SharedObject.getLocal()` och `SharedObject.getRemote()` för att använda ett lager med delade objekt som associeras med endast en del av URL-adressen. På så vis kan SWF-filen ge tillstånd till delning med andra SWF-filer från andra URL-adresser. Även om du skickar `'/'` som parametern `localPath` anges ändå ett lager med delade objekt som är specifikt för den egna domänen.

Användare kan begränsa åtkomst till delade objekt i dialogrutan Inställningar i Flash Player eller med Inställningshanteraren. Som standard kan delade objekt skapas med upp till 100 kB data per domän. Administratörer och användare kan också ange egna gränser för hur mycket som får skrivas i filsystemet. Mer information finns i avsnitten ”Administrativa inställningar” på sidan 1053 och ”Användarinställningar” på sidan 1055.

Du kan ange att ett delat objekt är säkert genom att ange parametern `true` för `secure` i metoden `SharedObject.getLocal()` eller `SharedObject.getRemote()`. Observera följande om parametern `secure`:

- Om den här parametern är inställd på `true` skapar Flash Player ett nytt säkert delat objekt eller hämtar en referens till ett befintligt säkert delat objekt. Det här säkra delade objektet kan bara läsas från eller skrivas till av SWF-filer som levereras över HTTPS och som anropar `SharedObject.getLocal()` med parametern `secure` inställd på `true`.
- Om den här parametern är inställd på `false` skapas ett nytt delat objekt eller så hämtas en referens till ett befintligt delat objekt som kan läsas från eller skrivas till av SWF-filer som levereras över icke-HTTPS-anslutningar.

Om anropande SWF-fil inte kommer från en HTTPS-URL genereras ett SecurityError-undantag om du anger `true` för parametern `secure` i metoden `SharedObject.getLocal()` eller `SharedObject.getRemote()`.

Valet av ett lager med delade objekt baseras på en SWF-fils ursprungliga URL-adress. Detta gäller även i de båda fallen när en SWF-fil inte kommer från en enkel URL-adress: import av inläsning och dynamisk inläsning. Import av inläsning avser de fall när du läser in en SWF-fil med egenskapen `LoaderContext.securityDomain` angiven med `SecurityDomain.currentDomain`. I det här fallet kommer den inlästa SWF-filen att ha en pseudo-URL-adress som inleds med inläsande SWF-filens domän och anger sedan den faktiska URL-adressen. Dynamisk inläsning avser inläsning av en SWF-fil med metoden `Loader.loadBytes()`. I det här fallet kommer den inlästa SWF-filen att ha en pseudo-URL-adress som inleds med inläsande SWF-fils hela URL-adress åtföljt av ett ID-tal. Vid både import av inläsning och dynamisk inläsning kan en SWF fils pseudo-URL-adress granskas med egenskapen `LoaderInfo.url`. Pseudo-URL-adressen behandlas på exakt samma sätt som en verklig URL-adress vad gäller val av lager med delade objekt. Du kan ange parametern `localPath` för delat objekt som använder delar av eller hela pseudo-URL-adressen.

Användare och administratörer kan välja att avaktivera användning av *utomstående objekt*. Detta innebär användning av delade objekt via en SWF-fil som körs i en webbläsare, när SWF-filens ursprungliga URL-adress är från en annan domän än URL:en som visas i webbläsarfältet. Användare och administratörer kan avaktivera utomstående objekt av säkerhetsskäl och för att undvika spårning över domäner. För att undvika den här begränsningen kanske du behöver vara säker på att SWF-filer som använder delade objekt läses in endast inom HTML-sidesstrukturer där SWF-filen garanterat kommer från samma domän som visas i webbläsarens adressfält. Om du försöker använda delade objekt från en utomstående SWF-fil, och användning av utomstående objekt har avaktiverats, returneras `null` från metoderna `SharedObject.getLocal()` och `SharedObject.getRemote()`. Mer information finns på www.adobe.com/products/flashplayer/articles/thirdpartylso.

Åtkomst av kamera, mikrofon, urklipp, mus och tangentbord

Flash Player 9 och senare, Adobe AIR 1.0 och senare

Om en SWF-fil försöker få åtkomst till en användares kamera eller mikrofon via metoderna `Camera.get()` eller `Microphone.get()` visas dialogrutan Sekretess där användaren kan tillåta eller neka åtkomst till kameran och mikrofonen. Användaren och administratören kan även avaktivera kameraåtkomst per plats eller globalt, via inställningar i filen mms.cfg, inställningsgränssnittet och inställningshanteraren (se ”Administrativa inställningar” på sidan 1053 och ”Användarinställningar” på sidan 1055). Med användarbegränsningar returneras värdet `null` för metoderna `Camera.get()` och `Microphone.get()` Du kan använda egenskapen `Capabilities.avHardwareDisable` när du ska avgöra om kameran och mikrofonen har blivit administrativt otillåtna (`true`) eller tillåtna (`false`).

Metoden `System.setClipboard()` tillåter en SWF-fil att ersätta innehållet i urklippet med en vanlig textsträng. Detta innebär ingen säkerhetsrisk. Det finns ingen motsvarande `getClipboard()`-metod som skyddar mot risken att lösenord och andra känsliga data klipps ut eller kopieras till urklipp.

Ett program som körs i Flash Player kan övervaka endast tangentbords- och mushändelser som sker inom dess fokus. Innehåll som körs i Flash Player kan inte identifiera tangentbords- eller mushändelser i andra program.

AIR-säkerhet

Adobe AIR 1.0 och senare

Grunderna i AIR-säkerhet

Adobe AIR 1.0 och senare

AIR-program körs med samma säkerhetsbegränsningar som systemprogram. I allmänhet har AIR-program, liksom systemprogram, en bred åtkomst till operativsystemfunktioner som att läsa och skriva filer, starta program, rita till skärmen och kommunicera med nätverket. Operativsystemsbegränsningar som gäller systemprogram, som till exempel användarspecifik behörighet, gäller också AIR-program.

Trots att säkerhetsmodellen i Adobe® AIR® är en utveckling av säkerhetsmodellen i Adobe® Flash® Player, skiljer sig säkerhetskontraktet från det säkerhetskontrakt som gäller för innehåll i en webbläsare. Avtalet ger utvecklare ett säkert medel för bredare funktionalitet för högfunktionella upplevelser med friheter som inte skulle vara lämpliga i ett webbläsarbaserat program.

AIR-program skrivs med antingen kompilerad bytekod (SWF-innehåll) eller tolkad skript (JavaScript, HTML) så att körtiden tillhandahåller minneshantering. Detta minimerar riskerna att AIR-program ska påverkas av brister i minneshanteringen, som till exempel buffertspill eller korrupt minne. Detta är några av de vanligaste svagheterna som påverkar datorprogram skrivna i inbyggd kod.

Installation och uppdateringar

Adobe AIR 1.0 och senare

AIR-program distribueras via AIR-installationsfiler (med filtillägget `air`) eller med inbyggda installationsprogram, som använder systemplattformens filformat och filtillägg. Installationsformatet för Windows är till exempel en EXE-fil, och för Android är det en APK-fil.

När Adobe AIR har installerats och en AIR-installationsfil öppnas hanteras installationsprocessen av körningsmiljön. När ett inbyggt installationsprogram används hanteras installationsprocessen av operativsystemet.

Obs! *Utvecklare kan ange en version, ett programnamn och en utgivarkälla med AIR-filformatet, men det går inte att ändra det inledande arbetsflödet för programinstallation. Denna begränsning är till fördel för användare eftersom samtliga AIR-program delar en säker, stömlinjeformad och konsekvent installationsprocedur som administreras av körtiden. Om programanpassning är nödvändig, kan den genomföras när programmet startas första gången.*

Installationsplats för körtiden

Adobe AIR 1.0 och senare

AIR-program med AIR-filformatet kräver att körningsmiljön ska vara installerad på datorn först, precis som SWF-filer kräver att webbläsarplugin-programmet för Flash Player ska vara installerat.

Körningsmiljön installeras på följande plats på stationära datorer:

- Mac OS: `/Library/Frameworks/`
- Windows®: `C:\Program Files\Common Files\Adobe AIR`
- Linux: `/opt/Adobe AIR/`

För att installera en uppdaterad version av ett program i Mac OS måste användaren ha tillräcklig behörighet för att installera i programkatalogen. I Windows och Linux måste användaren ha administrativ behörighet.

Obs! *I iOS installeras AIR-miljön inte separat; varje AIR-program är ett självständigt program.*

Körningsmiljön kan installeras på två sätt: med funktionen för sömlös installation (installera direkt från en webbläsare) eller via en manuell installation. AIR-program som paketeras som systemspecifika installationsprogram kan också installera AIR-miljön som en del av den normala programinstallationen. (Om du vill distribuera AIR-miljön på det här sättet måste du ha ett återdistributionsavtal med Adobe.)

Sömlös installation (körningsmiljö och program)

Adobe AIR 1.0 och senare

Funktionen för sömlös installation ger utvecklare tillgång till en strömlinjeformad installation för användare som ännu inte har installerat Adobe AIR. I den sömlösa installationsmetoden, skapar utvecklaren en SWF-fil som visar programmet för installation. När en användare klickar i SWF-filen för att installera programmet, görs ett försök att hitta körtiden. Om det inte går att hitta körtiden, installeras den. Därefter aktiveras körtiden omedelbart med installationsprocessen för utvecklarens program.

Manuell installation

Adobe AIR 1.0 och senare

Användaren kan också hämta och installera körtiden manuellt innan en AIR-fil öppnas. Utvecklaren kan sedan distribuera en AIR-fil på olika sätt (till exempel via e-post eller en HTML-länk på en webbsida). När AIR-filen öppnas, börjar körtiden bearbeta programinstallationen.

Programinstallationsflöde

Adobe AIR 1.0 och senare

Säkerhetsmodellen i AIR gör att användare kan bestämma om de ska installera ett AIR-program. AIR-installationen innebär flera förbättringar över inbyggda installationstekniker, vilket gör förtroendevalet enklare för användare:

- Körtiden tillhandahåller en konsekvent installation på alla operativsystem, även när ett AIR-program installeras från en länk i en webbläsare. De flesta inbyggda installationerna är beroende av webbläsaren eller ett annat program för att visa säkerhetsinformation, om det finns sådan över huvud taget.
- Installationen för ett AIR-program identifierar programkällan och vilka rättigheter som finns tillgängliga för programmet (om användaren tillåter att installationen fortsätter).
- Körtiden administrerar installationsprocessen för ett AIR-program. Det går inte att med ett AIR-program manipulera den installationsprocess som körtiden använder.

I regel ska en användare inte installera ett skrivbordsprogram som kommer från en källa som användaren inte litar på, eller från en som inte går att verifiera. Bevisbördan för säkerhet för inbyggda program är lika sann för AIR-program som den är för andra installerbara program.

Programmål

Adobe AIR 1.0 och senare

Installationskatalogen kan ställas in med ett av följande två alternativ:

1. Användaren anpassar målet under installationen. Programmet installeras där användaren önskar.

2 Om användaren inte ändrar installationsmålet, installeras programmet i standardsökvägen som bestäms i körtiden:
   - Mac OS: `~/Applications/`
   - Windows XP och tidigare: `C:\Program`
   - Windows Vista: `~/Apps/`
   - Linux: /opt/

   Om utvecklaren anger en inställning för `installFolder` i programmets beskrivningsfil, installeras programmet i en undersökväg till denna katalog.

AIR-filsystemet

Adobe AIR 1.0 och senare

Installationsprocessen för AIR-program kopierar alla filer som utvecklaren har inkluderat inuti AIR-installationsfilen till användarens dator. Det installerade programmet består av:

- Windows: En katalog med alla filer som inkluderades i AIR-installationsfilen. Körtiden skapar också en exe-fil under installationen av AIR-programmet.
- Linux: En katalog med alla filer som ingår i AIR-installationsfilen. Runtime-modulen skapar även en bin-fil under installationen av AIR-programmet.
- Mac OS: En `app`-fil som innehåller allt innehåll i AIR-installationsfilen. Filen kan undersökas genom användning av alternativet ”Visa paketinnehåll” i Finder. Körtiden skapar app-filen som en del av installationen av AIR-programmet.

Ett AIR-program körs genom att:

- Windows: Du kör .exe-filen i installationsmappen, eller kör en genväg som motsvarar .exe-filen (som till exempel en genväg på Start-menyn eller på skrivbordet).
- Linux: Starta bin-filen i installationsmappen, välj programmet i programmenyn eller kör från ett alias eller en skrivbordsgenväg.
- Mac OS: Du kör .app-filen eller ett alias som pekar på den.

Programmets filsystem innehåller också underkataloger som är relaterade till programmets funktion. Till exempel: Information som skrivs till krypterad lokal lagring sparas i en underkatalog i en katalog som namnges efter programmets programidentifierare.

AIR-programlagring

Adobe AIR 1.0 och senare

AIR-program har rättigheter för att skriva till valfri plats på hårddisken, men utvecklare uppmuntras till att använda sökvägen `app-storage:/` för lokal lagring relaterad till programmet. Filer som skrivs till `app-storage:/` från ett program lagras i en standardplats beroende på användarens operativsystem.

- I Mac OS: lagringskatalogen för programmet varierar mellan olika AIR-versioner:
   - **AIR 3.2 och tidigare** - `<appData>/<appId>/Local Store/` där `<appData>` är användarens "inställningsmapp", vanligtvis: `/Users/<user>/Library/Preferences`
   - **AIR 3.3 och senare** - `<path>/Library/Application Support/<appID>/Local Store`, där `<path>` antingen är `/Users/<user>/Library/Containers/<bundle-id>/Data` (i sandlådemiljö) eller `/Users/<user>` (vid körning utanför sandlådemiljö)

- I Windows: lagringskatalogen för ett program är `<appData>\<appId>\Local Store\`, där `<appData>` är användarens specialmapp CSIDL_APPDATA, vanligtvis: `C:\Documents and Settings\<användare>\Application Data`
- I Linux: `<appData>/<appID>/Local Store/`, där `<appData>` är `/home/<användare>/.appdata`

Du kan komma åt programlagringskatalogen via egenskapen `air.File.applicationStorageDirectory`. Du kan komma åt dess innehåll genom att använda metoden `resolvePath()` för klassen File. Detaljer finns under ”Arbeta med filsystemet” på sidan 655.

Uppdatera Adobe AIR

Adobe AIR 1.0 och senare

När användaren installerar ett AIR-program som kräver en uppdaterad version av körtiden, installeras körtidsuppdateringen automatiskt.

Användaren måste ha administrationsbehörighet för datorn för att kunna uppdatera körtiden.

Uppdatera AIR-program

Adobe AIR 1.0 och senare

Utveckling och distribution av programvaruuppdateringar är ett av de största säkerhetsproblemen för program skrivna med inbyggd kod. API:t för AIR innehåller en mekanism för att förbättra detta: Metoden `Updater.update()` kan anropas vid körning för att kontrollera en fjärrplats för en AIR-fil. Om det är lämpligt att uppdatera, hämtas och installeras AIR-filen, varpå programmet startas om. Utvecklare kan använda denna klass för att tillhandahålla nya funktioner och för att svara på potentiella svagheter i säkerheten.

Klassen Updater kan bara användas för att uppdatera program som distribueras som AIR-filer. Program som distribueras som inbyggda program måste använda operativsystemets uppdateringsfunktion, om en sådan finns.

Obs! *Utvecklare kan ange programmets version genom att ställa in egenskapen versionNumber i programmets beskrivningsfil.*

Avinstallera ett AIR-program

Adobe AIR 1.0 och senare

Om du tar bort ett AIR-program tar du också bort alla filer i programkatalogen. Filer som programmet kan ha skrivit till utanför programkatalogen tas dock inte bort. Även om du tar bort ett AIR-program tas inte de ändringar som AIR-programmet har gjort i filer utanför programkatalogen bort.

Inställningar i Windows-registret för administratörer

Adobe AIR 1.0 och senare

I Windows kan administratörer konfigurera en dator för att förhindra (eller tillåta) installation av AIR-program och körtidsuppdateringar. Dessa inställningar görs i Windows-registret under följande nyckel: HKLM\Software\Policies\Adobe\AIR. De kan vara några av följande:

Registerinställning	Beskrivning
AppInstallDisabled	Anger att installation och avinstallation av AIR-program är tillåtet. Ange 0 för ”tillåtet” och 1 för ”ej tillåtet”.
UntrustedAppInstallDisabled	Anger att installation av icke tillförlitliga AIR-program (program som inte omfattar ett betrott certifikat) är tillåten. Ange 0 för ”tillåtet” och 1 för ”ej tillåtet”.
UpdateDisabled	Anger att det är tillåtet att uppdatera körtiden, antingen som en bakgrundsuppgift eller som en del av en explicit installation. Ange 0 för ”tillåtet” och 1 för ”ej tillåtet”.

HTML-säkerhet i Adobe AIR

Adobe AIR 1.0 och senare

I det här avsnittet beskrivs AIR-arkitekturens HTML-säkerhet och hur du använder iframes, bildrutor och sandlådebryggan för att konfigurera HTML-baserade program och integrera HTML-innehåll i SWF-baserade program på ett säkert sätt.

Körtiden använder regler och innehåller mekanismer för att övervinna eventuella säkerhetssvagheter i HTML och JavaScript. Samma regler används oavsett om ditt program i huvudsak är skrivet i JavaScript eller om du läser in HTML- och JavaScript-innehållet till ett SWF-baserat program. Innehåll i programsandlådan och i icke-programsäkerhetssandlådan har olika rättigheter. När innehåll läses in till en bildruta eller iframe tillhandahåller körningsmiljön en säker funktion för *sandlådebryggor*, som tillåter innehåll i aktuell bildruta eller iframe att kommunicera säkert med innehåll i programsäkerhetssandlådan.

AIR SDK har tre klasser för återgivning av HTML-innehåll.

Klassen HTMLLoader ger en nära integrering mellan JavaScript-kod och API:erna i AIR.

Klassen StageWebView är en klass för HTML-återgivning och har mycket begränsad integrering med AIR-värdprogrammet. Innehåll som läses in av klassen StageWebView placeras aldrig i programmets säkerhetssandlåda och kan inte komma åt data eller anropa funktioner i AIR-värdprogrammet. På datorplattformar använder klassen StageWebView den inbyggda HTML-motorn i AIR, som bygger på Webkit, vilken också används av klassen HTMLLoader. På mobilplattformar använder klassen StageWebView operativsystemets HTML-kontroll. Därför har klassen StageWebView på mobilplattformar samma säkerhetsaspekter och -problem som systemets webbläsare.

Klassen TextField kan visa strängar med HTML-text. Ingen JavaScript-kod kan köras, men texten kan innehålla länkar och externa bilder.

Mer information finns under ”Undvika säkerhetsrelaterade JavaScript-fel” på sidan 987.

Översikt över att konfigurera ditt HTML-baserade program

Adobe AIR 1.0 och senare

Frames och iframes är ett bekvämt sätt att strukturera och organisera HTML-innehåll i AIR. Frames är ett sätt att både behålla databeständighet och att arbeta säkert med fjärrinnehåll.

Eftersom HTML i AIR bevarar sin normala sidbaserade organisering, kommer HTML-miljön att uppdateras fullständigt om HTML-innehållets huvud-frame öppnar en annan sida. Du kan använda frames och iframes för att behålla databeständighet i AIR, på samma sätt som du skulle göra för ett webbprogram som körs i en webbläsare. Om du definierar dina huvudsakliga programobjekt i programmets huvud-frame, kommer de att finnas kvar så länge du inte låter programmets huvud-frame gå till en ny sida. Använd underordnade frames eller iframes för att läsa in och visa programmets tillfälliga delar. (Det finns ett flertal sätt att behålla databeständighet som kan användas tillsammans med eller istället för frames. De är bland annat cookies, lokala delade objekt, lokal fillagring, den krypterade fillagringsplatsen och lokal databaslagring.)

Eftersom gränsen mellan körbar kod och data i HTML är suddig även i AIR, placeras innehåll i den översta bildrutan i HTML-miljön i programsandlådan. Efter `load`-händelsen för sidan begränsar AIR alla åtgärder, t.ex. `eval()`, som kan konvertera en sträng med text till ett körbart objekt. Den här begränsningen används till och med när ett program inte läser in fjärrinnehåll. Om du vill tillåta att HTML-innehåll kör dessa begränsade åtgärder måste du använda frames eller iframes för att placera innehållet i en icke-programsandlåda. (Du kan behöva köra innehåll i en underordnad bildruta som placerats i en sandlåda när du använder vissa JavaScript-programramverk som använder funktionen `eval()`.) En fullständig lista över begränsningar för JavaScript i programsandlådan finns under ”Kodrestriktioner för innehåll i olika sandlådor” på sidan 1087.

Eftersom HTML i AIR behåller sin förmåga att läsa in potentiellt osäkert fjärrinnehåll, använder AIR en samma-ursprung-policy som förhindrar att innehåll i en domän interagerar med innehåll i en annan. För att tillåta interaktion mellan programinnehåll och innehåll i en annan domän, kan du konfigurera en brygga som fungerar som gränssnittet mellan en huvud-frame och en underordnad frame.

Konfigurera en överordnad-underordnad sandlåderelation

Adobe AIR 1.0 och senare

AIR lägger till attributen `sandboxRoot` och `documentRoot` till frame- och iframe-elementen för HTML. Med dessa attribut kan du behandla programinnehåll som om det kom från en annan domän:

Attribut	Beskrivning
sandboxRoot	Den URL som används för att avgöra viken sandlåda och vilken domän som frame-innehåll ska placeras i. URL-schemat `file:`, `http:` eller `https:` måste användas.
documentRoot	Den URL från vilken frame-innehåll ska läsas in. URL-schemat `file:`, `app:` eller `app-storage:` måste användas.

I följande exempel mappas innehåll som är installerat i programmets sandlådeunderkatalog så att det körs i fjärrsandlådan och domänen www.example.com:

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

Konfigurera en brygga mellan huvud-frames och underordnade frames i olika sandlådor eller domäner

Adobe AIR 1.0 och senare

AIR lägger till egenskaperna `childSandboxBridge` och `parentSandboxBridge` till objektet `window` i alla underordnade frames. Med dessa egenskaper kan du definiera bryggor som fungerar som ett gränssnitt mellan en överordnad och en underordnad frame. Varje brygga går i en riktning:

`childSandboxBridge` - Egenskapen `childSandboxBridge` gör att en underordnad frame kan visa ett gränssnitt till innehåll i sin överordnade frame. Visa ett gränssnitt genom att ange egenskapen `childSandbox` till en funktion eller ett objekt i en underordnad frame. Sedan kan du komma åt objektet eller funktionen från innehåll i dess överordnade frame. Följande exempel visar hur ett skript som körs i en underordnad frame kan visa ett objekt som innehåller en funktion och en egenskap för dess överordnade frame.

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Om detta underordnade innehåll finns i en iframe som är tilldelad ett `id"child"`, kan du komma åt gränssnittet från överordnat innehåll genom att läsa frame-egenskapen `childSandboxBridge`.

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

`parentSandboxBridge` - Egenskapen `parentSandboxBridge` gör att en överordnad frame kan visa ett gränssnitt till innehåll i sin underordnade frame. Visa ett gränssnitt genom att ange egenskapen `parentSandbox` i en underordnad frame till en funktion eller ett objekt i dess överordnade frame. Sedan kan du komma åt objektet eller funktionen från innehåll i dess underordnade frame. Följande exempel visar hur ett skript som körs i en överordnad frame kan visa ett objekt som innehåller en spara-funktion för sin underordnade frame:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

Med detta gränssnitt kan innehåll i den underordnade framen spara text till en fil som heter save.txt. Den skulle dock inte ha någon annan tillgång till filsystemet. I allmänhet bör programinnehåll visa ett så litet gränssnitt som möjligt för andra sandlådor. Det underordnade innehållet kan anropa save-funktionen på följande sätt:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

Om underordnat innehåll försöker ange en egenskap för objektet `parentSandboxBridge`, utlöser körtiden ett SecurityError-undantag. Om överordnat innehåll försöker ange en egenskap för objektet `childSandboxBridge`, utlöser körtiden ett SecurityError-undantag.

Kodrestriktioner för innehåll i olika sandlådor

Adobe AIR 1.0 och senare

Som tidigare nämnts i inledningen till det här avsnittet, ”HTML-säkerhet i Adobe AIR” på sidan 1085, innehåller körningsmiljön regler och mekanismer för att hantera eventuella säkerhetsbrister i HTML och JavaScript. Detta avsnitt listar dessa begränsningar. Om kod försöker anropa dessa begränsade API:er utlöser körtiden ett fel med meddelandet ”Säkerhetsöverträdelse i Adobe AIR-körtiden för JavaScript-kod i programsäkerhetssandlådan”.

Mer information finns under ”Undvika säkerhetsrelaterade JavaScript-fel” på sidan 987.

Begränsningar för användning av JavaScript-funktionen eval() och liknande tekniker

Adobe AIR 1.0 och senare

Det finns begränsningar för HTML-innehåll i programsäkerhetssandlådan för att använda API:er som kan omvandla strängar dynamiskt till körbar kod efter att koden har lästs in (efter att händelsen `onload` i `body`-elementet har skickats och körningen av `onload`-hanterarfunktionen är slutförd). Detta är till för att förhindra programmet från att av misstag föra in (och köra) kod från källor som inte tillhör programmet (som till exempel potentiellt osäkra nätverksdomäner).

Om ditt program till exempel använder strängdata från en fjärrkälla för att skriva till egenskapen innerHTML i ett DOM-element, skulle strängen kunna innehålla körbar (JavaScript) kod som skulle kunna utföra osäkra operationer. Det är dock ingen risk att fjärrsträngar förs in i DOM medan innehållet laddas.

En begränsning är användandet av JavaScript-funktionen `eval()`. När kod i programsandlådan väl är inläst och efter bearbetning av onload-händelsehanteraren, kan du endast använda funktionen `eval()` på begränsade sätt. Följande regler gäller för användande av funktionen `eval()` *efter* att kod är inläst från programsäkerhetssandlådan:

- Uttryck som innefattar litteraler är tillåtna. Till exempel:

```
eval("null");
eval("3 + .14");
eval("'foo'");
```

- Objektlitteraler är tillåtna, som i följande:

```
{ prop1: val1, prop2: val2 }
```

- Set-/get-metoder för objektlitteraler är *ej tillåtna*, som i följande:

```
{ get prop1() { ... }, set prop1(v) { ... } }
```

- Arraylitteraler är tillåtna, som i följande:

```
[ val1, val2, val3 ]
```

- Uttryck som innefattar egenskapsläsningar är *ej tillåtna*, som i följande:

```
a.b.c
```

- Funktionsanrop är *ej tillåtna.*
- Funktionsdefinitioner är *ej tillåtna.*
- Inställning av egenskaper är *ej tillåten.*
- Funktionslitteraler är *ej tillåtna.*

Medan koden läses in, innan `onload`-händelsen och under körningen av `onload`-händelsehanterarfunktionen, gäller dock dessa begränsningar inte för innehåll i programsäkerhetssandlådan.

Efter att exempelvis kod har lästs in, resulterar följande kod i att körtiden utlöser ett undantag:

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

Dynamiskt genererad kod som till exempel den som tillverkas när funktionen `eval()` anropas skulle utgöra en säkerhetsrisk om den var tillåten inuti programsandlådan. Ett program kan till exempel av misstag köra en sträng som är inläst från en nätverksdomän, och den strängen kan innehålla uppsåtlig kod. Detta skulle till exempel kunna vara kod för att ta bort eller ändra filer på användarens dator. Det skulle också kunna vara kod som rapporterar innehållet i en lokal fil till en icke tillförlitlig nätverksdomän.

Följande sätt att generera dynamisk kod finns:

- Anropa funktionen `eval()`.
- Använda `innerHTML`-egenskaper eller DOM-funktioner för att infoga script-taggar som läser in ett skript utanför programkatalogen.
- Använda `innerHTML`-egenskaper eller DOM-funktioner för att infoga script-taggar som har intern kod (istället för att läsa in ett skript via attributet `src`).
- Ställa in attributet `src` för en `script`-tagg till att läsa in en JavaScript-fil som ligger utanför programkatalogen.
- Använda URL-schemat `javascript` (som i `href="javascript:alert('Test')"`).
- Använda funktionen `setInterval()` eller `setTimout()` där den första parametern (som definierar att funktionen ska köras asynkront) är en sträng (som ska utvärderas) istället för ett funktionsnamn (som i `setTimeout('x = 4', 1000)`).
- Anropa `document.write()` eller `document.writeln()`.

Kod som finns i programsäkerhetssandlådan kan bara använda dessa metoder medan innehåll läses in.

Dessa begränsningar förhindrar *inte* användandet av `eval()` med JSON-objektlitteraler. Detta gör att ditt programinnehåll kan arbeta med JavaScript-biblioteket JSON. Du hindras dock från att använda överlagrad JSON-kod (med händelsehanterare).

Om du vill använda andra Ajax-ramverk och JavaScript-kodbibliotek, bör du kontrollera om kod i ramverket eller biblioteket fungerar inom dessa restriktioner för dynamiskt genererad kod. Om de inte fungerar, ska du inkludera allt innehåll som använder ramverket eller biblioteket i en icke-programsäkerhetssandlåda. Mer information finns i ”Begränsningar för JavaScript inne i AIR” på sidan 1051 och ”Skriptning mellan program- och icke-programinnehåll” på sidan 1095. Adobe har en lista över Ajax-ramverk som stöder programsäkerhetssandlådan på http://www.adobe.com/products/air/develop/ajax/features/.

Till skillnad från innehåll i programsäkerhetssandlådan, *kan* JavaScript-innehåll i en icke-programsäkerhetssandlåda anropa funktionen `eval()` för att när som helst köra dynamiskt genererad kod.

Begränsningar i tillgång till AIR-API:er (för icke-programsandlådor)

Adobe AIR 1.0 och senare

JavaScript-kod i en icke-programsandlåda har inte tillgång till objektet `window.runtime`, och kan därför inte köra AIR-API:er. Om innehåll i en icke-programsäkerhetssandlåda anropar följande kod, utlöser programmet ett TypeError-undantag:

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

Undantagstypen är TypeError (odefinierat värde) eftersom innehållet i icke-programsandlådan inte känner igen objektet `window.runtime`, vilket gör att det ses som ett odefinierat värde.

Du kan visa körtidsfunktioner för innehåll i en icke-programsandlåda genom att använda en skriptbrygga. Fler detaljer finns under ”Skriptning mellan program- och icke-programinnehåll” på sidan 1095.

Begränsningar i att använda XMLHttpRequest-anrop

Adobe AIR 1.0 och senare

HTML-innehåll i programsäkerhetssandlådan kan inte använda synkrona XMLHttpRequest-metoder för att läsa in data som ligger utanför programsandlådan medan HTML-innehållet läses in och under `onLoad`-händelsen.

Som standard tillåts inte HTML-innehåll i icke-programsäkerhetssandlådor att använda JavaScript-objektet XMLHttpRequest för att läsa in data från andra domäner än den domän som anropar begäran. En `frame`- eller `iframe`-tagg kan innehålla ett `allowcrosscomainxhr`-attribut. Om det här attributet ställs in på ett värde som inte är null kan innehållet i bildrutan eller iframe använda JavaScript-objektet XMLHttpRequest för att läsa in data från andra domäner än den anropande kodens domän:

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

Mer information finns under ”Skriptning mellan innehåll i olika domäner” på sidan 1091.

Begränsningar för att läsa in CSS-, frame-, iframe- och img-element (för innehåll i icke-programsandlådor)

Adobe AIR 1.0 och senare

HTML-innehåll i fjärrsäkerhetssandlådor (nätverk) kan bara läsa in CSS-, `frame`-, `iframe`- och `img`-innehåll från fjärrsandlådor (från nätverks-URL:er).

HTML-innehåll i sandlådor av typerna lokal-med-filsystem, lokal-med-nätverk eller lokal-tillförlitlig, kan endast läsa in CSS-, frame-, iframe- och `img`-innehåll från lokala sandlådor (inte från program- eller fjärrsandlådor).

Begränsningar för anrop till JavaScript-metoden window.open()

Adobe AIR 1.0 och senare

Om ett fönster som skapas via ett anrop till JavaScript-metoden `window.open()` visar innehåll från en icke-programsäkerhetssandlåda, börjar fönstrets titel med huvudfönstrets (det skapande fönstret) titel, följt av tecknet kolon. Du kan inte använda kod för att flytta den delen av fönstertiteln utanför skärmen.

Innehåll i en icke-programsäkerhetssandlåda kan endast anropa JavaScript-metoden `window.open()` som svar på en händelse som triggas av en användarinteraktion med musen eller tangentbordet. Detta förhindrar att icke-programinnehåll skapar fönster som kan användas bedrägligt (till exempel för nätfiskeattacker). Händelsehanteraren för en mus- eller tangentbordshändelse kan inte heller ställa in metoden `window.open()` till att köras efter en fördröjning (genom att till exempel anropa funktionen `setTimeout()`).

Innehåll i fjärrsandlådor (nätverk) kan bara använda metoden `window.open()` för att öppna innehåll i fjärrnätverkssandlådor. Det kan inte använda metoden `window.open()` för att öppna innehåll från programsandlådan eller lokala sandlådor.

Innehåll i sandlådorna lokal-med-filsystem, lokal-med-nätverk eller lokal-tillförlitlig (se ”Säkerhetssandlådor” på sidan 1049) kan bara använda metoden `window.open()` för att öppna innehåll i lokala sandlådor. Det kan inte använda metoden `window.open()` för att öppna innehåll från programsandlådan eller fjärrsandlådor.

Fel vid anrop till begränsad kod

Adobe AIR 1.0 och senare

Om du anropar kod som är begränsad för användning i en sandlåda på grund av dessa säkerhetsbegränsningar, genererar körtiden ett JavaScript-fel: ”Säkerhetsöverträdelse i Adobe AIR-körtiden för JavaScript-kod i programsäkerhetssandlådan.”

Mer information finns under ”Undvika säkerhetsrelaterade JavaScript-fel” på sidan 987.

Sandlådeskydd vid inläsning av HTML-innehåll från en sträng

Adobe AIR 1.0 och senare

Med HTMLLoader-klassens `loadString()`-metod kan du skapa HTML-innehåll vid körning. Informationen som du använder som HTML-innehåll kan emellertid vara skadlig om den kommer från en osäker Internetkälla. Av den anledningen placeras inte HTML-innehåll som skapats med `loadString()`-metoden i programsandlådan som standard, och innehållet har inte åtkomst till AIR-API:erna. Du kan emellertid ändra värdet för `placeLoadStringContentInApplicationSandbox`-egenskapen för ett HTMLLoader-objekt till true om du vill att HTML-innehåll som skapats med `loadString()`-metoden ska placeras i programsandlådan. Mer information finns i avsnittet ”Läsa in HTML-innehåll från en sträng” på sidan 986.

Skriptning mellan innehåll i olika domäner

Adobe AIR 1.0 och senare

AIR-program ges särskilda behörigheter när de installeras. Det är mycket viktigt att dessa rättigheter inte läcks till annat innehåll, inklusive fjärrfiler och lokala filer som inte är en del av programmet.

Om sandlådebryggan i AIR

Adobe AIR 1.0 och senare

Normalt sett kan inte innehåll från andra domäner anropa skript i andra domäner. För att förhindra att AIR-program av misstag läcker konfidentiell information eller kontroll, placeras följande begränsningar på innehåll i `program`säkerhetssandlådan (innehåll som installeras med programmet):

- Kod i programsäkerhetssandlådan kan inte skapa tillåtelser för andra sandlådor genom att anropa metoden `Security.allowDomain()`. Om den här metoden anropas från programsandlådan returneras ett fel.
- Det går inte att importera icke-programinnehåll till programsandlådan genom att ange egenskaperna `LoaderContext.securityDomain` eller `LoaderContext.applicationDomain`.

Det finns fortfarande fall där huvud-AIR-programmet kräver att innehåll från en fjärrdomän ska ha kontrollerad tillgång till skript i huvud-AIR-programmet, eller vice versa. För att uppnå detta innehåller körtiden en *sandlådebrygga*-mekanism som fungerar som en brygga mellan de båda sandlådorna. En sandlådebrygga kan skapa explicit interaktion mellan fjärr- och programsäkerhetssandlådor.

Sandlådebryggan visar två objekt som både skript som är inlästa och skript som läses in kan komma åt:

- Med objektet `parentSandboxBridge` kan innehåll som läses in visa egenskaper och funktioner för det inlästa innehållet.
- Med objektet `childSandboxBridge` kan inläst innehåll visa egenskaper och funktioner för skript i det innehåll som läses in.

Objekt som visas via sandlådebryggan skickas genom värde, och inte referens. Alla data är serialiserade. Detta betyder att de objekt som visas av en sida av bryggan inte kan ställas in från den andra sidan, och att de visade objekten är typlösa. Du kan bara visa enkla objekt och funktioner. Du kan inte visa komplexa objekt.

Om underordnat innehåll försöker ange en egenskap för objektet parentSandboxBridge, utlöser körtiden ett SecurityError-undantag. På samma sätt gäller att om överordnat innehåll försöker ange en egenskap för objektet childSandboxBridge, utlöser körtiden ett SecurityError-undantag.

Exempel på sandlådebrygga (SWF)

Adobe AIR 1.0 och senare

Anta att ett AIR-program i form av en musikbutik vill tillåta fjärr-SWF-filer att sända priset på ett album, men att fjärr-SWF-filen inte ska visa om priset är ett reapris eller ej. För att göra detta innehåller en StoreAPI-klass en metod för att hämta priset, men dölja reapriset. En instans av klassen StoreAPI tilldelas sedan egenskapen `parentSandboxBridge` för LoaderInfo-objektet för det Loader-objekt som läser in fjärr-SWF:en.

Det följande är koden för AIR-musikbutiken:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
title="Music Store" creationComplete="initApp()">
    <mx:Script>
        import flash.display.Loader;
        import flash.net.URLRequest;

        private var child:Loader;
        private var isSale:Boolean = false;

        private function initApp():void {
            var request:URLRequest =
                    new URLRequest("http://[www.yourdomain.com]/PriceQuoter.swf")

            child = new Loader();
            child.contentLoaderInfo.parentSandboxBridge = new StoreAPI(this);
            child.load(request);
            container.addChild(child);
        }
        public function getRegularAlbumPrice():String {
            return "$11.99";
        }
        public function getSaleAlbumPrice():String {
            return "$9.99";
        }
        public function getAlbumPrice():String {
            if(isSale) {
                return getSaleAlbumPrice();
            }
            else {
                return getRegularAlbumPrice();
            }
        }
    </mx:Script>
    <mx:UIComponent id="container" />
</mx:WindowedApplication>
```

StoreAPI-objektet anropar huvudprogrammet för att hämta standardpriset på albumet, men returnerar ”Inte tillgänglig” när metoden `getSaleAlbumPrice()` anropas. Följande kod definierar klassen StoreAPI:

```
public class StoreAPI
{
    private static var musicStore:Object;

    public function StoreAPI(musicStore:Object)
    {
        this.musicStore = musicStore;
    }

    public function getRegularAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }

    public function getSaleAlbumPrice():String {
        return "Not available";
    }

    public function getAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }
}
```

Följande kod representerar ett exempel på en PriceQuoter-SWF-fil som rapporterar butikens pris, men som inte kan rapportera reapriset:

```
package
{
    import flash.display.Sprite;
    import flash.system.Security;
    import flash.text.*;

    public class PriceQuoter extends Sprite
    {
        private var storeRequester:Object;

        public function PriceQuoter() {
            trace("Initializing child SWF");
            trace("Child sandbox: " + Security.sandboxType);
            storeRequester = loaderInfo.parentSandboxBridge;

            var tf:TextField = new TextField();
            tf.autoSize = TextFieldAutoSize.LEFT;
            addChild(tf);

            tf.appendText("Store price of album is: " + storeRequester.getAlbumPrice());
            tf.appendText("\n");
            tf.appendText("Sale price of album is: " + storeRequester.getSaleAlbumPrice());
        }
    }
}
```

Exempel på sandlådebrygga (HTML)

Adobe AIR 1.0 och senare

För HTML-innehåll läggs egenskaperna `parentSandboxBridge` och `childSandboxBridge` till i JavaScript-fönsterobjektet för ett underordnat dokument. Ett exempel på hur du ska konfigurera bryggfunktioner i HTML-innehåll finns under ”Konfigurera ett gränssnitt för en sandlådebrygga” på sidan 1005.

Begränsa API-exponering

Adobe AIR 1.0 och senare

När du visar sandlådebryggor är det viktigt att visa högnivå-API:er som begränsar den grad som de kan missbrukas på. Kom ihåg att det innehåll som anropar bryggimplementeringen kan vara komprometterat (genom exempelvis kodinjektion). Därför är visning av en `readFile(path:String)`-metod (som läser innehållet i en godtycklig fil) via en brygga känsligt för missbruk. Det skulle vara bättra att visa ett `readApplicationSetting()`-API som inte innehåller en sökväg, och som läser en specifik fil. Den mer semantiska inställningen begränsar den skada som ett program kan göra om delar av det blir komprometterade.

Fler hjälpavsnitt

”Korsskriptning av innehåll i olika säkerhetssandlådor” på sidan 1003

”AIR-programsandlådan” på sidan 1050

Skriva till disk

Adobe AIR 1.0 och senare

Program som körs i en webbläsare har endast begränsad interaktion med användarens lokala filsystem. Webbläsare använder säkerhetspolicyer som säkerställer att en användares dator inte kan kompromettеras som ett resultat av inläsning av webbinnehåll. Till exempel: SWF-filer som körs genom Flash Player i en webbläsare kan inte interagera direkt med filer som redan finns på en användares dator. Delade objekt och cookies kan skrivas till användarens dator i syfte att behålla användarinställningar och andra data, men det är gränsen för interaktionen med filsystemet. Eftersom AIR-program installeras på datorn, har de ett annorlunda säkerhetskontrakt som innefattar möjligheten att läsa och skriva i det lokala filsystemet.

Denna frihet medför ett stort ansvar för utvecklare. Oavsiktliga programosäkerheter äventyrar inte bara programmets funktion, utan också integriteten för användarens dator. Av den anledningen bör utvecklare läsa ”Bästa säkerhetsrutiner för utvecklare” på sidan 1096.

AIR-utvecklare kan komma åt och skriva filer i det lokala filsystemet med hjälp av flera olika URL-schemakonventioner:

URL-schema	Beskrivning
app:/	Ett alias för programkatalogen. Filer som öppnas från den här sökvägen tilldelas till programsandlådan och har de fullständiga privilegierna som kan ges av körtiden.
app-storage:/	Ett alias för den lokala lagringskatalogen, som standardiseras i körtiden. Filer som öppnas från den här sökvägen tilldelas en icke-programsandlåda.
file:///	Ett alias som representerar roten på användarens hårddisk. En fil som öppnas från den här sökvägen tilldelas en programsandlåda om filen finns i programkatalogen, och annars till en icke-programsandlåda.

Obs! *AIR-program kan inte modifiera innehåll genom att använda URL-schemat app:. Programkatalogen kan också vara skrivskyddad på grund av administratörsinställningar.*

Förutsatt att det inte finns några adminstratörsinställda begränsningar på användarens dator, kan AIR-program skriva var som helst på användarens hårddisk. Utvecklare rekommenderas att använda sökvägen `app-storage:/` för lokal lagring som är relaterad till deras program. Filer som skrivs till `app-storage:/` från ett program placeras i en standardplats:

- I Mac OS: lagringskatalogen för ett program är `<appData>/<appId>/Local Store/` där `<appData>` är användarens inställningsmapp. Detta är vanligtvis `/Users/<user>/Library/Preferences`
- I Windows: lagringskatalogen för ett program är `<appData>\<appId>\Local Store\` där `<appData>` är användarens specialmapp CSIDL_APPDATA. Detta är vanligtvis `C:\Documents and Settings\<userName>\Application Data`
- I Linux: `<appData>/<appID>/Local Store/`, där `<appData>` är `/home/<användare>/.appdata`

Om ett program är utformat för att interagera med befintliga filer i användarens filsystem ska du läsa ”Bästa säkerhetsrutiner för utvecklare” på sidan 1096.

Arbeta säkert med icke tillförlitligt innehåll

Adobe AIR 1.0 och senare

Innehåll som inte är tilldelat programsandlådan kan innehålla ytterligare skriptningsfunktioner för programmet, men bara om det möter körtidens säkerhetskriterier. Detta avsnitt förklarar säkerhetskontraktet med icke-programinnehåll i AIR.

Security.allowDomain()

Adobe AIR 1.0 och senare

AIR-program begränsar skriptningsåtkomst för icke-programinnehåll mer stringent än vad webbläsar-plug-inen Flash Player begränsar skriptning för icke tillförlitligt innehåll. När en SWF-fil som associeras med sandlådan `lokal-tillförlitlig` anropar metoden `System.allowDomain()` i Flash Player i webbläsaren beviljas exempelvis skriptåtkomst till allt SWF-innehåll som lästs in från den angivna domänen. Motsvarande metod är inte tillåten från `program`innehåll i AIR-program, eftersom detta skulle ge en orimlig åtkomst till användarens filsystem för icke-programfilen. Fjärrfiler kan inte komma åt programsandlådan direkt, även om det finns anrop till metoden `Security.allowDomain()`.

Skriptning mellan program- och icke-programinnehåll

Adobe AIR 1.0 och senare

AIR-program som skriptar mellan program- och icke-programinnehåll har mer invecklade säkerhetsarrangemang. Filer som inte ligger i programsandlådan tillåts endast åtkomst till egenskaper och metoder för filer i programsandlådan genom användandet av en sandlådebrygga. En sandlådebrygga fungerar som en brygga mellan programinnehåll och icke-programinnehåll, vilket ger explicit interaktion mellan de två filerna. När de används på rätt sätt, innebär sandlådebryggor ett extra säkerhetslager som begränsar åtkomsten till objektreferenser som är del av programinnehållet för icke-programinnehåll.

Fördelarna med sandlådebryggor visas bäst genom exempel. Anta att ett AIR-musikbutikprogram vill ge ett API till annonsörer som vill skapa egna SWF-filer som butiksprogrammet sedan kan kommunicera med. Butiken vill ge annonsörerna metoder för att leta upp artister och skivor i butiken, men vill också av säkerhetsskäl isolera några metoder och egenskaper från SWF-filerna från tredje part.

En sandlådebrygga kan innehålla denna funktion. Som standard har innehåll som laddas in i AIR-programmet externt vid körningen inte åtkomst till några metoder eller egenskaper i huvudprogrammet. Med en anpassad sandlådebryggimplementering kan en utvecklare tillhandahålla tjänster till fjärrinnehållet utan att visa dessa metoder eller egenskaper. Se på sandlådebryggan som en stig mellan tillförlitligt och icke-tillförlitligt innehåll, som tillhandahåller kommunikation mellan inläsar- och utläsarinnehåll utan att visa objektreferenser.

Mer information om hur du använder sandlådebryggor på ett säkert sätt finns under ”Skriptning mellan innehåll i olika domäner” på sidan 1091.

Skydd mot att dynamiskt generera osäkert SWF-innehåll

Adobe AIR 1.0 och senare

Med metoden `Loader.loadBytes()` kan ett program generera SWF-innehåll från en bytearray. Injektionsattacker på data som är inlästa från fjärrkällor skulle dock kunna orsaka stor skada under inläsning av innehåll. Detta är särskilt sant under inläsning av data till programsandlådan där det genererade SWF-innehållet kan komma åt den fulla uppsättningen AIR-API:er.

Det finns befogade användningar för att nyttja metoden `loadBytes()` utan att generera körbar SWF-kod. Du kan till exempel använda metoden `loadBytes()` till att generera bilddata för att kontrollera tiden för bildvisning. Det finns också befogade användningar som *förlitar* sig på att köra kod, som till exempel dynamiskt skapande av SWF-innehåll för ljuduppspelning. Som standard kan metoden `loadBytes()` i AIR *inte* läsa in SWF-innehåll. Den kan bara läsa in bildinnehåll. I AIR har egenskapen `loaderContext` för metoden `loadBytes()` en `allowLoadBytesCodeExecution`-egenskap som du kan ange som `true` för att ge explicit tillåtelse till programmet att använda `loadBytes()` för att läsa in körbart SWF-innehåll. Följande kod visar hur funktionen används:

```
var loader:Loader = new Loader();
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.allowLoadBytesCodeExecution = true;
loader.loadBytes(bytes, loaderContext);
```

Om du anropar `loadBytes()` för att läsa in SWF-innehåll och egenskapen `allowLoadBytesCodeExecution` för LoaderContext-objektet är satt till `false` (standarden), utlöser Loader-objektet ett SecurityError-undantag.

Obs! *Detta API kan komma att ändras i en framtida version av Adobe AIR. När det inträffar, kan du behöva kompilera om innehåll som använder egenskapen `allowLoadBytesCodeExecution` för klassen LoaderContext.*

Bästa säkerhetsrutiner för utvecklare

Adobe AIR 1.0 och senare

Även om AIR-program byggs med hjälp av webbtekniker, är det viktigt att utvecklare observerar att de inte arbetar inuti webbläsarsäkerhetssandlådan. Detta innebär att det är möjligt att bygga AIR-program som kan skada det lokala systemet, antingen avsiktligt eller oavsiktligt. AIR försöker att minimera risken, men det finns fortfarande sätt att introducera svagheter. Detta avsnitt täcker viktiga potentiella osäkerheter.

Risk vid import av filer till programsäkerhetssandlådan

Adobe AIR 1.0 och senare

Filer som finns i programkatalogen tilldelas programsandlådan och har körtidens fullständiga behörigheter. Program som skriver till det lokala filsystemet rekommenderas att skriva till `app-storage:/`. Denna katalog finns separat från programfilerna på användarens dator. Filerna tilldelas således inte programsandlådan och innebär därför en reducerad säkerhetsrisk. Utvecklare bör tänka på följande:

- Inkludera endast en fil i en AIR-fil (i det installerade programmet) om det är nödvändigt.
- Inkludera endast en skriptningsfil i en AIR-fil (i det installerade programmet) om du till fullo förstår dess beteende och om den är tillförlitlig.
- Skriv inte till, eller modifiera inte, innehåll i programkatalogen. Körtiden förhindrar att program skriver eller modifierar filer och kataloger med URL-schemat `app:/` genom att utlösa ett SecurityError-undantag.
- Använd inte data från en nätverkskälla som parametrar till AIR-API-metoder som kan leda till kodkörning. Detta inkluderar användning av metoden `Loader.loadBytes()` och JavaScript-funktionen `eval()`.

Risk vid användning av en fjärrkälla för att bestämma sökvägar

Adobe AIR 1.0 och senare

Ett AIR-program kan kompromettеras när det använder fjärrdata eller -innehåll. Av den anledningen bör du vara särskilt försiktig när du använder data från nätverket eller filsystemet. Tillförlitlighetsbördan ligger ytterst på utvecklaren och de nätverksanslutningar som han eller hon skapar, men det medför alltid en risk att läsa in externa data, och de bör inte användas som indata till känsliga operationer. Utvecklare uppmanas att undvika följande:

- Använda data från en nätverkskälla för att bestämma ett filnamn
- Använda data från en nätverkskälla för att konstruera en URL som programmet använder för att skicka privat information

Risk vid användning, lagring eller sändning av osäkra uppgifter

Adobe AIR 1.0 och senare

Att lagra användaruppgifter i det lokala filsystemet introducerar alltid risken att dessa uppgifter kan kompromettеras. Utvecklare bör tänka på följande:

- Kryptera autentiseringsuppgifterna som skrivs till det lokala filsystemet om de ska lagras lokalt. Körtiden innehåller en krypterad lagring som är unik för varje installerat program genom klassen EncryptedLocalStore. Läs mer i avsnittet ”Krypterad lokal lagringsplats” på sidan 712.
- Skicka inte okrypterade användaruppgifter till en nätverkskälla såvida inte källan är tillförlitlig och protokollen HTTPS: eller TLS (Transport Layer Security) används vid överföringen.
- Specificera aldrig ett standardlösenord vid uppgiftsskapande - låt användarna skapa egna. användare som inte ändrar standardinställningen exponerar sina autentiseringsuppgifter för angripare som redan känner till det förvalda lösenordet.

Risk vid en nedgraderingsattack

Adobe AIR 1.0 och senare

Under programinstallation kontrollerar körtiden att det inte redan finns en installerad version av programmet. Om ett program redan är installerat, jämför körtiden versionssträngen mot den version som installeras. Om strängen är en annan kan användare välja att uppgradera installationen. Körtiden garanterar inte att den nyinstallerade versionen är nyare än den redan installerade versionen, bara att det är en annan version. En angripare kan distribuera en äldre version till användaren för att ta sig förbi en säkerhetssvaghet. Av den anledningen bör utvecklaren göra versionskontroller när programmet körs. Det är en god idé att låta program kontrollera nätverket efter nödvändiga uppdateringar. Även om en angripare får användaren att köra en gammal version, kommer den gamla versionen på så sätt att detektera att den behöver uppdateras. Användning av ett tydligt versionsschema för programmet gör också att det blir svårare att lura användare att installera en nedgraderad version.

Kodsignering

Adobe AIR 1.0 och senare

Alla AIR-installationsfiler måste vara kodsignerade. Kodsignering är en kryptografisk process för att bekräfta att det angivna programvaruursprunget är riktigt. AIR-program kan signeras med ett certifikat som utfärdats av en extern certifikatutfärdare (CA) eller med ett självsignerat certifikat som du själv skapar. Ett kommersiellt certifikat från en välkänd certifikatutfärdare rekommenderas. På så sätt kan dina användare lita på att det är ditt program de installerar, och inte en förfalskning. Du kan dock skapa ett självsignerat certifikat genom att använda `adt` från SDK eller genom att använda antingen Flash, Flash Builder eller ett annat program som använder `adt` för att generera certifikat. Självsignerade certifikat ger ingen försäkran om att programmet som installeras är äkta och bör inte användas för att testa ett program för allmän lansering.

Säkerhet på Android-enheter

Adobe AIR 2.5 och senare

På Android, liksom på alla datorenheter, följer AIR den inbyggda säkerhetsmodellen. Samtidigt har AIR egna säkerhetsregler, som är avsedda att underlätta för utvecklare att skriva säkra, Internet-anslutna program.

Eftersom AIR-program på Android använder Androids paketformat lyder installationen under Androids säkerhetsmodell. AIR-programmets installationsprogram används inte.

De tre huvudsakliga aspekterna i Android-säkerhetsmodellen är:

- Behörighet
- Programsignaturer
- Användar-ID för program

Android-behörigheter

Många Android-funktioner skyddas av operativsystemets behörighetsfunktioner. För att kunna använda en skyddad funktion måste AIR-programbeskrivningsfilen deklarera att programmet behöver den nödvändiga behörigheten. När en användare försöker installera programmet visar Android-operativsystemet alla begärda behörigheter för användaren innan installationen inleds.

De flesta AIR-program måste ange Android-behörigheter i programbeskrivningsfilen. Som standard ingår inga behörigheter. Följande behörigheter krävs för skyddade Android-funktioner som används av AIR-miljön:

ACCESS_COARSE_LOCATION Tillåter programmet att komma åt platsdata för WIFI och mobilnätverk via klassen Geolocation.

ACCESS_FINE_LOCATION Tillåter programmet att komma åt GPS-data via klassen Geolocation.

ACCESS_NETWORK_STATE och ACCESS_WIFI_STATE Tillåter programmet att komma åt nätverksinformation via klassen NetworkInfo.

CAMERA Tillåter programmet att komma åt kameran.

INTERNET Tillåter programmet att utföra nätverksbegäranden och fjärrfelsökning.

READ_PHONE_STATE Tillåter AIR-miljön att stänga av ljudet vid inkommande samtal.

RECORD_AUDIO Tillåter programmet att komma åt mikrofonen.

WAKE_LOCK och DISABLE_KEYGUARD Tillåter att programmet hindrar enheten från att försättas i viloläge via inställningarna för klassen SystemIdleMode.

WRITE_EXTERNAL_STORAGE Tillåter programmet att skriva till enhetens externa minneskort.

Programsignaturer

Alla programpaket som skapas för Android-plattformen måste signeras. Eftersom AIR-program på Android paketeras i Androids APK-format signeras de i enlighet med Androids regler i stället för de som gäller i AIR. Även om Android och AIR använder kodsignering på liknande sätt, finns det betydande skillnader:

- På Android verifierar signaturen att utvecklaren äger den privata nyckeln, men den används inte för att verifiera utvecklarens identitet.
- För program som ska lanseras för Android måste certifikatet vara giltigt i minst 25 år.
- Android saknar stöd för att migrera paketsignaturer till nya certifikat. Om en uppdatering signeras av ett annat certifikat måste användarna avinstallera det ursprungliga programmet innan de kan installera uppdateringen.
- Två program som signerats med samma certifikat kan ange ett delat ID, som tillåter dem att komma åt varandras cacheminnen och datafiler. (Sådan delning medges inte av AIR.)

Användar-ID för program

Android använder en Linux-kärna. Alla installerade program tilldelas ett användar-ID av Linux-typ, och detta avgör vilka behörigheter som gäller för till exempel filåtkomst. Filer i programmet, programkatalogen och tillfälliga kataloger åtkomstskyddas av filsystemets behörigheter. Filer som skrivs till externa lagringsplatser (t.ex. SD-kortet) kan läsas, ändras och tas bort av andra program, eller av användaren, när SD-kortet monteras som en masslagringsenhet på en dator.

Cookies som tas emot med Internet-begäranden delas inte mellan olika AIR-program.

Sekretess för bakgrundsbilder

När en användare lägger ett program i bakgrunden, tas i vissa Android-versioner en skärmdump som används som miniatyr i programlistan för nyligen använda program. Denna skärmdump lagras i enhetens minne och en angripare med fysik kontroll över enheten kan komma åt den.

Om känslig information visas i programmet ska du skydda sådan information som finns i skärmdumpen i bakgrunden. Händelsen `deactivate` som skickas av NativeApplication-objektet signalerar att ett program kommer att placeras i bakgrunden. Du bör använda denna händelse för att dölja eller rensa all känslig information.

Fler hjälpavsnitt

Android: Security and Permissions (Säkerhet och behörigheter)

Krypterade data på Android

AIR-program på Android kan använda de krypteringsalternativ som finns i den inbyggda SQL-databasen för att spara krypterade data. För att få optimal säkerhet ska du basera krypteringsnyckeln på ett användargivet lösenord som anges när programmet körs. En lokalt lagrad krypteringsnyckel eller ett lösenord är svårt eller omöjligt att "dölja" för en angripare som har åtkomst till programfilerna. Om angriparen får tag i nyckeln, erbjuder kryptering av data inget ytterligare skydd utöver den användar-ID-baserade filsystemssäkerheten som tillhandahålls i Android-systemet.

Klassen The EncryptedLocalStore kan användas för att spara data, men dessa data krypteras inte på Android-enheter. I stället förlitar sig säkerhetsmodellen i Android på programanvändar-ID för att skydda data från andra program. Program som använder ett delat användar-ID och som är signerade med samma kodsigneringscertifikat använder samma krypterade lokala lagerplats.

Viktigt! *På en "öppen" telefon har alla program som körs med rotprivilegier åtkomst till filer i andra program. Detta medför att data lagrade i den krypterade lokala lagerplatsen inte är skyddade på "öppna" enheter.*

Säkerhet på iOS-enheter

I iOS följer AIR den inbyggda säkerhetsmodellen. Samtidigt har AIR egna säkerhetsregler, som är avsedda att underlätta för utvecklare att skriva säkra, Internet-anslutna program.

Eftersom AIR-program på iOS använder iOS-paketformatet lyder installationen under iOS-säkerhetsmodellen. AIR-programmets installationsprogram används inte. Dessutom används ingen separat AIR-körningsmiljö på iOS-enheter. Alla AIR-program innehåller den kod de behöver för att fungera.

Programsignaturer

Alla programpaket som skapas för iOS-plattformen måste signeras. Eftersom AIR-program på iOS paketeras i iOS IPA-format signeras de i enlighet med iOS regler i stället för de som gäller i AIR. Även om iOS och AIR använder kodsignering på liknande sätt, finns det betydande skillnader:

- I iOS måste det certifikat som används för att signera ett program vara utfärdat av Apple. Certifikat från andra utfärdare kan inte användas.
- I iOS är de distributionscertifikat som utfärdas av Apple vanligtvis giltiga i ett år.

Sekretess för bakgrundsbilder

När en användare lägger ett program i bakgrunden i iOS, tar operativsystemet en skärmdump som används för att animera övergången. Denna skärmdump lagras i enhetens minne och en angripare med fysik kontroll över enheten kan komma åt den.

Om känslig information visas i programmet ska du skydda sådan information som finns i skärmdumpen i bakgrunden. Händelsen `deactivate` som skickas av NativeApplication-objektet signalerar att ett program kommer att placeras i bakgrunden. Du bör använda denna händelse för att dölja eller rensa all känslig information.

Kapitel 65: Använda ActionScript-exempel

Att köra ett exempel på en ActionScript 3.0-kod är det bästa sättet att lära sig hur speciella klasser och metoder fungerar. Du kan använda exempel på olika sätt, beroende på enheten som du använder eller riktar in dig mot.

Datorer som kör Flash Professional eller Flash Builder Mer information om dessa utvecklingsmiljöer för hur man använder ActionScript 3.0-exempel finns i ”Köra ActionScript 3.0-exempel i Flash Professional” på sidan 1103 och ”Köra ActionScript 3.0-exempel i Flash Builder” på sidan 1104. Använd trace-satser eller andra felsökningsverktyg för att öka förståelsen för hur ett kodexempel fungerar.

Mobila enheter Du kan köra ActionScript 3.0-kodexempel på mobila enheter som har stöd för Flash Player 10.1 och senare versioner. Mer information finns i ”Köra ActionScript 3.0-exempel på mobila enheter” på sidan 1105. Du kan även köra dessa exempel på datorn om du använder Flash Professional eller Flash Builder.

Tv-enheter Trots att du inte kan köra dessa exempel på tv-enheter, kan du fortfarande lära dig av exemplen genom att köra dem på datorn. Mer information om webbplatsen Adobe Developer Connection och hur du utvecklar program för tv-enheter finns på Flash Platform for TV

Exempeltyper

De olika typerna av ActionScript 3.0-kodexempel är:

- Exempel på kodfragment (finns genomgående i ActionScript 3.0-dokumentationen)
- Klassbaserade exempel (finns i första hand i ActionScript 3.0 språkreferens)
- Praktiska exempel med flera källfiler (hämta källfilerna i ZIP-format från www.adobe.com/go/learn_programmingAS3samples_flash_se)

Exempel på kodfragment

Ett exempel på kodfragment ser ut enligt följande:

```
var x:int = 5;
trace(x); // 5
```

Kodfragment innehåller endast tillräcklig kod för att demonstrera en enskild idé. De innehåller vanligtvis inte paket- eller klassuttryck.

Klassrelaterade exempel

Många exempel visar källkoden för en komplett ActionScript-klass. Ett klassrelaterat exempel ser ut enligt följande:

```
package {
    public class Example1 {
        public function Example1():void {
            var x:int = 5;
            trace(x); //5
        }
    }
}
```

Koden för ett klassrelaterat exempel innehåller ett paketuttryck, en klassförklaring och en konstruktionsfunktion.

Praktiska exempel med flera källfiler

Många av artiklarna i Utvecklarhandbok för ActionScript 3.0 avslutas med ett praktiskt exempel som visar hur en viss ActionScript-funktion kan användas i ett praktiskt, verkligt sammanhang. Exemplen innehåller vanligtvis flera filer:

- En eller flera ActionScript-källfiler
- En .FLA-fil för användning i Flash Professional
- En eller flera MXML-filer för användning i Flash Builder
- Datafiler, bildfiler, ljudfiler eller andra resurser som används i exempelprogrammet (tillval).

Praktiska exempel levereras normalt som arkivfiler i ZIP-format.

Lista med utvecklarhandboksexempel i ZIP-filen

ZIP-filen för Flash Professional CS5 och Flex 4 (hämtas från www.adobe.com/go/learn_programmingAS3samples_flash_se) innehåller följande exempel:

- AlarmClock (”Exempel på händelsehantering: väckarklocka” på sidan 141)
- AlgorithmicVisualGenerator (”Exempel på Drawing API: algoritmisk visuell generator” på sidan 233)
- ASCIIArt (”Exempel på strängar: ASCII-bilder” på sidan 19)
- CapabilitiesExplorer (”Exempel på egenskaper: identifera systemegenskaper” på sidan 876
- CustomErrors (”Exempel på att hantera fel: programmet CustomErrors” på sidan 69)
- DisplayObjectTransformer (”Exempel på geometri: tillämpa en matrisomformning på ett visningsobjekt” på sidan 217)
- FilterWorkbench (”Exempel på filtrering av visningsobjekt: Filter Workbench” på sidan 293)
- GlobalStockTicker (”Exempel: Internationalisering av ett börskursprogram” på sidan 957)
- IntrovertIM_HTML (”Exempel på extern API: kommunicera mellan ActionScript och JavaScript i en webbläsare” på sidan 851)
- NewsLayout (”Exempel på textfält: Textformatering i tidningsstil” på sidan 387)
- PlayList (”Exempel på array: PlayList” på sidan 47)
- PodcastPlayer (”Exempel på ljud: Podcast Player” på sidan 467)
- ProjectionDragger (”Exempel: Perspektivprojektion” på sidan 358)
- ReorderByZ (”Ändra visningsordning med hjälp av Matrix3D-objekt” på sidan 362)
- RSSViewer (”Exempel på XML i ActionScript: läsa in RSS-data från Internet” på sidan 113)
- RuntimeAssetsExplorer (”Exempel på filmklipp: RuntimeAssetsExplorer” på sidan 332)
- SimpleClock (”Exempel på datum och tid: enkel analog klocka” på sidan 6)
- SpinningMoon (”Exempel på bitmapp: animerad snurrande måne” på sidan 255)
- SpriteArranger (”Exempel på visningsobjekt: SpriteArranger” på sidan 203)
- TelnetSocket (”Exempel på TCP-socket: skapa en Telnet-klient” på sidan 801)
- VideoJukebox (”Exempel på video: videojukebox” på sidan 507)
- WikiEditor (”Exempel på reguljära uttryck: en Wiki-parser” på sidan 90)
- WordSearch (”Exempel på indata från mus: WordSearch” på sidan 578)

Praktiska exempel finns även i många av snabbstartsartiklarna i Flash Developer Center och Flex Developer Center.

Köra ActionScript 3.0-exempel i Flash Professional

Använd en av följande procedurer (vilken beror på exempeltyp) för att köra ett exempel med Flash Professional.

Köra ett kodfragment i Flash Professional

Så här kör du ett kodfragment i Flash Professional:

1. Välj Arkiv > Nytt.
2. I dialogrutan Nytt dokument väljer du Flash-dokument och klickar på OK.

   Ett nytt Flash-fönster visas.
3. Klicka på den första bildrutan i det första lagret i tidslinjepanelen.
4. Skriv eller klistra in kodfragmentexemplet på åtgärdspanelen.
5. Välj Arkiv > Spara. Ge filen ett namn och klicka på OK.
6. Välj Kontroll > Testa filmen om du vill testa exemplet.

Köra ett klassrelaterat exempel i Flash Professional

Så här kör du ett klassrelaterat exempel i Flash Professional:

1. Välj Arkiv > Nytt.
2. I dialogrutan Nytt dokument väljer du ActionScript-fil och klickar på OK. Ett nytt redigeringsfönster visas.
3. Kopiera den klassrelaterade exempelkoden och klistra in den i redigeringsfönstret.

   Om klassen är huvuddokumentklass för programmet måste den utöka klassen MovieClip:

   ```
   import flash.display.MovieClip;
   public class Example1 extends MovieClip{
   //...
   }
   ```

   Kontrollera även att alla klasser som exemplet refererar till deklareras med `import`-satser.
4. Välj Arkiv > Spara. Ge filen samma namn som klassen i exemplet (till exempel SammanhangMenyExempel.as).

   Obs! *Vissa av de klassbaserade exemplen, som flashx.textLayout.container.ContainerController, innehåller flera nivåer i paketdeklarationen (`package flashx.textLayout.container.examples {`). För de här exemplen sparar du filen i en undermapp som matchar paketdeklarationen (flashx/textLayout/container/examples) eller tar bort paketnamnet (så att ActionScript-koden börjar med bara `package { `), så kan du testa filen från vilken plats som helst.*
5. Välj Arkiv > Nytt.
6. I dialogrutan Nytt dokument väljer du Flash-dokument (ActionScript 3.0) och klickar på OK. Ett nytt Flash-fönster visas.
7. I panelen Egenskaper i fältet Dokumentklass anger du namnet på exempelklassen som bör motsvara namnet på källfilen i ActionScript som du just sparade (till exempel SammanhangMenyExempel).
8. Välj Arkiv > Spara. Ge FLA-filen samma namn som klassen i exemplet (till exempel SammanhangMenyExempel.fla).
9. Välj Kontroll > Testa filmen om du vill testa exemplet.

Köra ett praktiskt exempel i Flash Professional

Praktiska exempel levereras normalt som arkivfiler i ZIP-format. Så här kör du ett praktiskt exempel i Flash Professional:

1. Packa upp arkivfilen i en valfri mapp.
2. Välj Arkiv > Öppna i Flash Professional.
3. Bläddra till den mapp där du packade upp arkivfilen. Välj FLA-filen i den mappen och klicka på Öppna.
4. Välj Kontroll > Testa filmen om du vill testa exemplet.

Köra ActionScript 3.0-exempel i Flash Builder

Använd en av följande procedurer (vilken beror på exempeltyp) för att köra ett exempel med Flash Builder.

Köra ett kodfragment i Flash Builder

Så här kör du ett kodfragment i Flash Builder:

1. Skapa antingen ett nytt Flex-projekt (Arkiv > Nytt > Nytt Flex-projekt) eller skapa ett nytt MXML-program inom ett befintligt Flex-projekt (Arkiv > Nytt > MXML-program). Ge projektet eller programmet ett beskrivande namn (som SammanhangMenyExempel).
2. Lägg till en `<mx:Script>`-tagg i den genererade MXML-filen.
3. Klistra in innehållet i kodfragmentexemplet mellan taggarna `<mx:Script>` och `</mx:Script>`. Spara MXML-filen.
4. Om du vill köra exemplet väljer du Kör > Kör menyalternativ för MXML-filen (som Kör > Kör SammanhangMenyExempel).

Köra ett klassrelaterat exempel i Flash Builder

Så här kör du ett klassrelaterat exempel i Flash Builder:

1. Välj Arkiv > Nytt > ActionScript-projekt.
2. Ange namnet på den primära klassen (som SammanhangMenyExempel) i fältet Projektnamn. Använd standardvärden för andra fält (eller ändra dem enligt din specifika miljö). Klicka på Slutför om du vill skapa projektet i ActionScript-filen.
3. Radera allt genererat innehåll från ActionScript-filen. Klistra in exempelkoden, inklusive paket- och importuttryck, i ActionScript-filen och spara filen.

   Obs! *Vissa av de klassbaserade exemplen, som flashx.textLayout.container.ContainerController, innehåller flera nivåer i paketdeklarationen (`package flashx.textLayout.container.examples {`). För de här exemplen sparar du filen i en undermapp som matchar paketdeklarationen (flashx/textLayout/container/examples) eller tar bort paketnamnet (så att ActionScript-koden börjar med bara `package {`), så kan du testa filen från vilken plats som helst.*

4. Om du vill köra exemplet väljer du Kör > Kör menyalternativ för ActionScript-klassnamnet (som Kör > Kör SammanhangMenyExempel).

Köra ett praktiskt exempel i Flash Builder

Praktiska exempel levereras normalt som arkivfiler i ZIP-format. Så här kör du ett praktiskt exempel i Flash Builder:

1. Packa upp arkivfilen i en valfri mapp. Ge mappen ett beskrivande namn (som (SammanhangMenyExempel).

2 Välj Arkiv > Nytt Flex-projekt i Flash Builder. I avsnittet Projektplats klickar du på Bläddra och väljer mappen med exempelfilerna. I fältet Projektnamn anger du mappens namn (som (SammanhangMenyExempel). Använd standardvärden för andra fält (eller ändra dem enligt din specifika miljö). Klicka på Nästa om du vill fortsätta.

3 I panelen Utdata klickar du på Nästa om du vill acceptera standardvärdet.

4 I panelen Källsökvägar klickar du på knappen Bläddra intill fältet Huvudprogramfil. Välj MXML-exempelfilen från exempelmappen. Klicka på Slutför om du vill skapa projektfilerna.

5 Om du vill köra exemplet väljer du Kör > Kör menyalternativ för MXML-filen (som Kör > Kör SammanhangMenyExempel).

Köra ActionScript 3.0-exempel på mobila enheter

Du kan köra ActionScript 3.0-kodexempel på mobila enheter som stödjer Flash Player 10.1. Emellertid är det vanligast att köra kodexempel för att lära sig hur särskilda klasser och metoder fungerar. I så fall kör du exemplet på en icke-mobil enhet som en stationär dator. På den stationära datorn kan du använda spårningsuttryck och andra felsökningsverktyg i Flash Professional eller Flash Builder för att öka din förståelse av ett kodexempel.

Om du vill köra exemplet på en mobil enhet kan du antingen kopiera filerna till enheten eller till en webbserver. Om du vill kopiera filerna till enheten och köra exemplet i webbläsaren gör du följande:

1 Skapa en SWF-fil genom att följa instruktionerna i ”Köra ActionScript 3.0-exempel i Flash Professional” på sidan 1103 eller i ”Köra ActionScript 3.0-exempel i Flash Builder” på sidan 1104. I Flash Professional skapar du SWF-filer genom att välja Kontroll > Testa filmen. I Flash Builder skapar du SWF-filer genom att köra, felsöka eller bygga dina Flash Builder-projekt.

2 Kopiera SWF-filen till en katalog på den mobila enheten. Använd den programvara som följer med enheten för att kopiera filen.

3 Ange fil://-adress för SWF-filen i webbläsarens adressfält på den mobila enheten. Ange exempelvis `fil:://program/mittExempel.swf`

Om du vill kopiera filer till en webbserver och köra exemplet i enhetens webbläsare gör du följande:

1 Skapa en SWF-fil och en HTML-fil. Följ först instruktionerna i ”Köra ActionScript 3.0-exempel i Flash Professional” på sidan 1103 eller i ”Köra ActionScript 3.0-exempel i Flash Builder” på sidan 1104. I Flash Professional innebär Kontroll > Testa filmen att du skapar endast SWF-filen. Om du vill skapa bägge filerna väljer du först Flash och HTML på fliken Format i dialogrutan Publiceringsinställningar. Sedan väljer du Arkiv > Publicera för att skapa både HTML- och SWF-filer. I Flash Builder skapar du både SWF-filer och HTML-filer när du kör, felsöker eller bygger ditt Flash Builder-projekt.

2 Kopiera SWF-filen och HTML-filen till en katalog på webbservern.

3 Ange HTTP-adressen för HTML-filen i webbläsarens adressfält på den mobila enheten. Ange exempelvis `http://www.minWebbserver/exempel/mittExempel.html`

Innan du kör ett exempel på en mobil enhet bör du överväga följande problem.

Visningsstorlek

Visningsstorleken som används när du kör ett exempel på en mobil enhet är mycket mindre än när du använder en icke-mobil enhet. Många exempel kräver ingen särskild visningsstorlek. När du skapar SWF-filen anger du en visningsstorlek som är lämplig för din mobila enhet. Ange exempelvis 176 x 208 pixlar.

Syftet med de praktiska exemplen i Utvecklarhandbok för ActionScript 3.0 är att illustrera olika koncept och klasser i ActionScript 3.0. Deras användargränssnitt har utformats för att se bra ut och fungera väl på en stationär eller bärbar dator. Även om exemplen fungerar på mobila enheter är visningsstorleken och användargränssnittet inte lämpliga för små skärmar. Adobe rekommenderar att du kör det praktiska exemplet på en dator för att lära dig ActionScript, och sedan använda relevanta kodfragment i dina mobilprogram.

Textfält i stället för spåruttryck

När du kör ett exempel på en mobil enhet kan du inte se utdata från exemplets spåruttryck. Skapa en instans i stället för klassen TextField om du vill se utdata. Lägg sedan till texten från spåruttrycken till egenskapen `text` i textfältet.

Du kan använda följande funktion om du vill ställa in ett textfält för spårning:

```
function createTracingTextField(x:Number, y:Number,
                                width:Number, height:Number):TextField {

    var tracingTF:TextField = new TextField();
    tracingTF.x = x;
    tracingTF.y = y;
    tracingTF.width = width;
    tracingTF.height = height;

    // A border lets you more easily see the area the text field covers.
    tracingTF.border = true;
    // Left justifying means that the right side of the text field is automatically
    // resized if a line of text is wider than the width of the text field.
    // The bottom is also automatically resized if the number of lines of text
    // exceed the length of the text field.
    tracingTF.autoSize = TextFieldAutoSize.LEFT;

    // Use a text size that works well on the device.
    var myFormat:TextFormat = new TextFormat();
    myFormat.size = 18;
    tracingTF.defaultTextFormat = myFormat;

    addChild(tracingTF);
    return tracingTF;
}
```

Lägg exempelvis till funktionen till dokumentklassen som en privat funktion. Spåra sedan data i andra metoder i dokumentklassen med kod som följande:

```
var traceField:TextField = createTracingTextField(10, 10, 150, 150);
// Use the newline character "\n" to force the text to the next line.
traceField.appendText("data to trace\n");
traceField.appendText("more data to trace\n");
// Use the following line to clear the text field.
traceField.appendText("");
```

Med metoden `appendText()` accepteras endast ett värde som en parameter. Detta värde är en sträng (antingen en String-instans eller en stränglitteral). Om du vill skriva ut värdet för en variabel som inte är en sträng, måste du först konvertera värdet till ett String-värde. Det enklaste sättet att göra detta är att anropa objektets `toString()`-metod:

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

Textstorlek

I många textexempel används textfält för att illustrera ett koncept. Ibland medför en justering av textstorleken i textfältet bättre läsbarhet på en mobil enhet. Om exempelvis en textfältsinstans med namnet `myTextField` används i ett exempel ändras dess textstorlek med följande kod:

```
// Use a text size that works well on the device.
var myFormat:TextFormat = new TextFormat();
myFormat.size = 18;
myTextField.defaultTextFormat = myFormat
```

Samla in indata från användarna

Mobilens operativsystem och webbläsare samlar in vissa indata från användarna som inte tas emot av SWF-innehållet. Specifikt beteende beror på operativsystem och webbläsare men oväntat beteende kan förekomma när du kör exemplen på en mobil enhet. Läs mer i ”Prioritet för tangentbordshändelser” på sidan 562.

Vidare är användargränssnitten i många exempel utformade för stationära eller bärbara datorer. Exempelvis är de flesta av de praktiska exemplen i Utvecklarhandbok för ActionScript 3.0 lämpliga för visning på stationära datorer. Därför visas ibland inte helheten på den mobila enhetens skärm. Förmågan att panorera genom webbläsarens innehåll beror på webbläsaren. Vidare är exemplen inte utformade för att fånga eller hantera bläddrings- eller panoreringshändelser. Därför är vissa exempels användargränssnitt inte lämpliga att köra på en liten skärm. Adobe rekommenderar att du kör exemplen på en dator för att lära dig ActionScript, och sedan använda relevanta kodfragment i dina mobilprogram.

Läs mer i ”Panorering och rullning av visningsobjekt” på sidan 178.

Hantera fokus

Vissa exempel kräver att du ger ett fält fokus. Genom att ge ett fält fokus kan du exempelvis ange text eller välja en knapp. För att ge ett fält fokus använder du den mobila enhetens pekarenhet, exempelvis ett stift eller ditt finger. Använd annars den mobila enhetens navigationsknappar för att ge fältet fokus. Om du vill välja en knapp som har fokus använder du Välj-knappen på den mobila enheten, precis som du använder Retur på en dator. På vissa enheter måste du trycka på knappen två gånger för att välja den.

Läs mer om fokus i ”Hantera fokus” på sidan 557.

Hantera musåtgärder

Många åtgärder kräver musrörelser. På en dator kan de här musrörelserna inträffa exempelvis när en användare rullar över ett skärmobjekt med musen eller klickar med musknappen på ett skärmobjekt. Rörelser med pekarenheter som stift eller finger kallas pekrörelser på mobila enheter. I Flash Player 10.1 omvandlas pekrörelser till musrörelser. I den omvandlingen garanteras felfri funktion för SWF-innehåll som utvecklades före Flash Player 10.1. Därför fungerar exempel när en pekenhet används för att välja eller dra och visa objekt.

Prestanda

Mobila enheter har mindre bearbetningskraft än stationära enheter. Vissa CPU-krävande exempel uppvisar ibland sämre prestanda på mobila enheter. Till exempel utför exemplet i ”Exempel på Drawing API: algoritmisk visuell generator” på sidan 233 omfattande uträkningar och ritningar innan varje bildruta genereras. När exemplet körs på en dator illustreras andra ritnings-API:er. Emellertid är exemplet inte lämpligt på vissa mobila enheter på grund av begränsade prestanda.

Mer information om prestanda på mobilenheter finns i *Optimera prestanda för Flash-plattformen*.

God praxis

I exemplen tas inte hänsyn till god praxis i programutveckling för mobila enheter. Begränsningar i minne och bearbetningskraft i mobila enheter kräver särskild hänsyn. På liknande sätt finns i användargränssnittet för en liten skärm andra krav än för en stationär skärm. Mer information om hur du utvecklar program för mobilenheter *finns i Optimera prestanda för Flash-plattformen*.

Kapitel 66: SQL-stöd i lokala databaser

I Adobe AIR ingår en databasmotor med stöd för lokala SQL-databaser med många standardfunktioner som använder databassystemet SQLite, baserat på öppen källkod. Körningen specificerar inte hur eller var databasdata lagras i filsystemet. Varje databas lagras i en enda fil. En utvecklare kan specificera den plats i filsystemet där databasfilen lagras, och en AIR-applikation kan komma åt en eller flera separata databaser (d.v.s. separata databasfiler). I det här dokumentet beskrivs SQL-syntax och datatypsstöd för lokala SQL-databaser i Adobe Air. Det här dokumentet är inte tänkt att vara en heltäckande SQL-referens. Dokumentet beskriver snarare specifika detaljer av den SQL-dialekt som Adobe AIR stöder. Körtidsbiblioteket stöder huvuddelen av SQL-dialekten SQL-92. Eftersom det finns ett flertal referenser, webbplatser, böcker och utbildningsmaterial för att lära sig SQL är det här materialet inte menat att vara ett omfattande material om SQL eller en självstudiekurs. I stället läggs fokus i det här dokumentet på SQL-stödet i Apollo AIR, och skillnaderna mellan SQL-92 och den SQL-dialekt som stöds.

Konventioner för SQL-satsdefinitioner

Inom programsatsdefinitionerna i det här dokumentet används följande konventioner:

- Stora och små bokstäver
  - UPPER CASE – reserverade SQL-nyckelord skrivs med stora bokstäver.
  - lower case – platshållartermer eller satsnamn skrivs med små bokstäver.
- Definitionstecken
  - ::= – indikerar en sats eller en programsatsdefinition.
- Gruppering och ändring av tecken
  - | – lodstreck används mellan alternativ och ska utläsas som "eller".
  - [] - objekt inom hakparenteser är valfria objekt, parentesen innehåller ett objekt eller en uppsättning alternativa objekt
  - () – parenteser runt en uppsättning alternativ (där alternativen är separerade av lodstreck), anger en obligatorisk grupp av objekt, det vill säga en uppsättning av objekt som är möjliga värden för ett obligatoriskt objekt.
- Kvantifierare
  - + – ett plustecken följt av ett objekt inom parentes indikerar att föregående objekt kan förekomma en eller flera gånger.
  - * – en asterisk följt av ett objekt inom parentes indikerar att föregående objekt (inom parentes) kan förekomma 0 eller flera gånger.
- Literala tecken
  - * – en asterisk som används i ett kolumnnamn eller mellan parenteser följt av ett funktionsnamn innebär en literal asterisk snarare än "0 eller mer" kvantifierare.
  - . . – en punkt representerar en literal punkt.
  - , – ett kommatecken representerar ett literalt komma.
  - () – parentes runt en enda sats eller ett objekt indikerar att parentesen är obligatorisk och literal.
  - Andra tecken representerar literala tecken om inte annat sägs.

SQL-syntax som stöds

I det här avsnittet beskrivs den SQL-syntax som stöds av SQL-databasmotorn i Adobe AIR. Dessa är uppdelade i förklaringar i olika programsatser och satstyper, uttryck, inbyggda funktioner och operatorer. Följande ämnen finns med:

- Allmän SQL-syntax
- Programsatser för databehandling (SELECT, INSERT, UPDATE och DELETE)
- Programsatser för datadefinitioner (CREATE, ALTER och DROP för tabeller, index, vyer och utlösare)
- Särskilda programsatser och klausuler
- Inbyggda funktioner (sammanställning, skalärfunktioner och datum-/tid-formateringsfunktioner)
- Operatorer
- Parametrar
- SQL-funktioner som inte stöds
- Ytterligare SQL-funktioner

Allmän SQL-syntax

Förutom den specifika syntax som gäller för olika satser och uttryck, används följande generella regler för alla SQL-uttryck:

Skiftlägeskänslighet SQL-satser, inklusive objektnamn, är inte skiftlägeskänsliga. Däremot skrivs SQL-programsatser vanligtvis med SQL-nyckelord i versaler, så även i det här dokumentet. SQL-uttryck är inte skifteslägeskänsliga, men det är däremot literaltextvärden i SQL. Dessutom kan jämförelser och sorteringsåtgärder vara skifteslägesberoende, enligt vad som specificeras av den kollationssekvens som är definierad för en kolumn eller åtgärd. Mer information finns i COLLATE.

Blanktecken Ett blanktecken (som blanksteg, tabb, nyradstecken o.s.v.) måste användas för att avgränsa individuella ord i en SQL-programsats. Däremot är blanktecken mellan ord och symboler valfritt. Vilken typ av och hur många blanktecken du använder har ingen betydelse i SQL-uttryck. Du kan använda blanktecken, t.ex. indrag och radbrytningar, för att formatera dina SQL-satser så att de blir enklare att läsa, utan att det påverkar satsens innehåll.

Programsatser för databehandling

Programsatser för databehandling är de vanligaste SQL-programsatserna. Dessa programsatser används för att hämta, lägga till, ändra och ta bort data från databastabeller. Följande programsatser för databehandling stöds: SELECT, INSERT, UPDATE, och DELETE.

SELECT

Programsatsen SELECT används för att köra databasfrågor. Resultatet av en SELECT-sats är noll eller flera rader med data, där varje rad har ett fast antal kolumner. Antal kolumner i resultatet specificeras i resultatkolumnnamnet eller i uttryckslistan mellan SELECT och valfria FROM-nyckelord.

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

Vilket godtyckligt uttryck som helst kan användas som ett resultat. Om ett resultatuttryck är * ersätts alla kolumner i tabellen med det uttrycket. Om uttrycket är namnet på en tabell följt av *, är resultatet alla kolumner i just den tabellen.

Nyckelordet DISTINCT gör att en delmängd av resultatraderna returneras, där alla raderna är unika. NULL-värden behandlas inte som om de skiljer sig från varandra. Standardbeteendet är att alla resultatrader skickas tillbaka, vilket kan förtydligas med nyckelordet ALL.

Databasfrågan körs mot en eller flera tabeller som anges efter nyckelordet FROM. Om flera tabellnamn avgränsas med kommatecken returneras alla poster i de olika tabellerna med en korskoppling (cross join). JOIN-syntaxen kan också användas för att specificera hur tabellerna kopplas. Den enda typen av yttre koppling (outer join) som stöds är LEFT OUTER JOIN. Satsuttrycket ON i join-args måste kunna evalueras som ett booleskt värde. En underfråga inom parentes kan användas som en tabell i FROM-satsen. Hela FROM-satsen kan utelämnas och i så fall blir resultatet en enkel rad bestående av värdena i resultatuttryckslistan.

Satsuttrycket WHERE används för att begränsa det antal rader som frågan hämtar. WHERE-satsuttrycken måste kunna evalueras som ett booleskt värde. WHERE-satsfiltreringen utförs före gruppering, vilket innebär att WHERE-satsuttrycken inte kan innehålla sammanställningsfunktioner.

Satsen GROUP BY gör att en eller flera resultatrader kombineras till en enda utdatarad. En GROUP BY-sats är speciellt användbar om resultatet innehåller sammanställningsfunktioner. Uttrycken i GROUP BY-satsen behöver inte vara uttryck som visas i SELECT-uttryckslistan.

HAVING-satsen fungerar på samma sätt som WHERE i den bemärkelsen att den begränsar det antal rader som returneras av programsatsen. HAVING-satsen tillämpas däremot efter att en gruppering som har specificerats av GROUP BY har skett. Följaktligen kan HAVING-uttrycket referera till värden som inkluderar sammanställningsfunktioner. Det behöver inte finnas ett HAVING-satsuttryck i SELECT-listan. Precis som ett WHERE-uttryck måste ett HAVING-uttryck kunna resultera i ett booleskt värde.

ORDER BY-satsen sorterar de utgående raderna. Argumentet sort-expr-list för ORDER BY-satsen är en lista med uttryck som används som nyckel för sorteringen. Uttrycken behöver inte vara del av resultatet för en enkel SELECT, men i en sammansatt SELECT (en SELECT använder en av compound-op-operatorerna) måste varje uttryck exakt matcha en av resultatkolumnerna. Varje sorteringsuttryck kan följas av en sort-order-sats bestående av COLLATE-nyckelord och namnet på en kollationsfunktion som används för att sortera text och/eller nyckelordet ASC eller DESC för att specificera sorteringsordning (stigande eller fallande). sort-order kan utelämnas, och då används standardordningen (stigande ordning). En definition av COLLATE-satsen och kollationsfunktioner finns i COLLATE.

LIMIT-satsen sätter en övre gräns för de antal rader som returneras i resultatet. Ett negativt LIMIT-värde betyder att det inte finns någon övre gräns. OFFSET efter LIMIT används för att specificera hur många rader som ska hoppas över i början av resultatet. I en sammansatt SELECT-fråga kan LIMIT-satsen endast visas efter den slutliga SELECT-programsatsen och begränsningen gäller då för hela frågan. Observera, att om nyckelordet OFFSET används i LIMIT-satsen, är gränsen det första heltalet och förskjutningen det andra heltalet. Om ett komma används i stället för nyckelordet OFFSET, är förskjutningen den första siffran och gränsen den andra siffran. Den här inkonsekvensen är avsiktlig, eftersom den maximerar kompatibiliteten med äldre SQL-databassystem.

En sammansatt SELECT är uppbyggd av två eller flera enkla SELECT-satser som förenas med någon av operatorerna UNION, UNION ALL, INTERSECT eller EXCEPT. I en sammansatt SELECT måste alla SELECT-satserna specificera samma antal resultatkolumner. Det får bara finnas en ORDER BY-sats efter den slutliga SELECT-programsatsen (och före LIMIT-satsen, om den finns med). Operatorerna UNION och UNION ALL kombinerar resultatet från föregående och efterföljande SELECT-satser i en enda tabell. Skillnaden är, att i UNION är alla resultatrader distinkta, till skillnad från UNION ALL där det kan finnas dubbletter. Operatorn INTERSECT ger snittet av den föregående och efterföljande SELECT-programsatsen. EXCEPT ger resultatet av föregående SELECT efter att ha tagit bort resultatet av efterföljande SELECT. Om tre eller flera SELECT-programsatser är sammankopplade, grupperas de från första till sista resultatmängd.

En definition av tillåtna uttryck finns i Uttryck.

Från och med AIR 2.5 stöds SQL CAST-operatorn vid läsning för att konvertera BLOB-data till ActionScript ByteArray-objekt. Följande kod läser till exempel rådata som inte sparas i AMF-format och sparar dessa i ett ByteArray-objekt:

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

INSERT

Programsatsen INSERT finns i två grundläggande former och används för att lägga till data i tabeller.

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] VALUES (value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] select-statement
               REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                    REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

Den första formen (med nyckelordet VALUES) skapar en ny rad i en befintlig tabell. Om ingen column-list har angetts, måste antalet värden vara detsamma som antalet kolumner i tabellen. Om en column-list har angetts, måste antalet värden matcha antalet specificerade kolumner. Kolumner i tabellen som inte finns med i kolumnlistan fylls med de standardvärden som definierades när tabellen skapades, eller med NULL om inget standardvärde är definierat.

Den andra formen av INSERT hämtar data från en SELECT-programsats. Antal kolumner i resultatet av SELECT-uttrycket måste exakt matcha antalet kolumner i tabellen, om column-list inte har angetts, eller så måste det matcha antalet kolumner som är namngivna i kolumnlistan. En ny post skapas i tabellen för varje resultatrad från SELECT-uttrycket. SELECT-uttrycket kan vara enkelt eller sammansatt. En definition av tillåtna SELECT-programsatser finns i SELECT.

Det valfria uttrycket conflict-algorithm tillåter en specifikation av en alternativ begränsad konfliktlösningsalgoritm som kan användas under detta kommando. Du hittar förklaringar och definitioner av konfliktalgoritmer i avsnittet ”Särskilda programsatser och klausuler” på sidan 1119.

De två REPLACE INTO-formerna av programsatsen motsvarar att använda INSERT [OR conflict-algorithm]-standardformen med konfliktalgoritmen REPLACE (d.v.s. INSERT OR REPLACE...).

De två REPLACE INTO-formerna av programsatsen motsvarar att använda INSERT [OR conflict-algorithm]-standardformen med konfliktalgoritmen REPLACE (d.v.s. INSERT OR REPLACE...).

UPDATE

Uppdateringskommandot ändrar de befintliga posterna i en tabell.

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,...
[WHERE expr]
```

Kommandot består av nyckelordet UPDATE följt av namnet på den tabell vars poster ska uppdateras. Efter nyckelordet SET anger du namnet på kolumnen och det värde som kolumnen ska ändras till som en kommaavgränsad lista. WHERE-satsuttrycket tillhandahåller den eller de rader i vilka poster uppdateras.

DELETE

Kommandot DELETE används för att ta bort poster ur en tabell.

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

Kommandot består av nyckelordet DELETE FROM följt av namnet på den tabell från vilken posterna ska tas bort.

Utan en WHERE-sats tas alla rader i tabellen bort. Om en WHERE-sats finns med, tas endast rader som matchar uttrycket bort. WHERE-satsuttrycken måste kunna evalueras som ett booleskt värde. En definition av tillåtna uttryck finns i Uttryck.

Programsatser för datadefinitioner

Programsatser för definitioner av data används för att skapa, ändra och ta bort databasobjekt som tabeller, vyer, utlösare och index. Följande programsatser för definitioner av data stöds:

- Tabeller:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- Index:
  - CREATE INDEX
  - DROP INDEX
- Vyer:
  - CREATE VIEWS
  - DROP VIEWS
- Utlösare:
  - CREATE TRIGGERS
  - DROP TRIGGERS

CREATE TABLE

En CREATE TABLE-programsats består av nyckelorden CREATE TABLE följt av namnet på den nya tabellen och sedan (inom parentes) en lista på definitioner och begränsningar för kolumnerna. Tabellnamnen kan antingen vara en identifierare eller en sträng.

```
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-
name
                          ( column-def [, column-def]* [, constraint]* )
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-
statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value     ::= NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

Varje kolumndefinition är namnet på kolumnen följt av kolumnens datatyp och sedan en eller flera valfria kolumnbegränsningar. Kolumnens datatyp bestämmer vilka data som kan lagras i den kolumnen. Om det görs ett försök att lagra ett värde i en kolumn med en annan datatyp, kommer körningsbiblioteket att konvertera värdet till lämplig typ, om detta är möjligt. Går det inte att konvertera värdet genereras ett fel. Mer information finns i avsnittet Datatyper som stöds.

Kolumnbegränsningen NOT NULL anger att kolumnen inte får innehålla NULL-värden.

En UNIQUE-begränsning innebär att ett index skapas för en eller flera kolumner. Det här indexet måste innehålla unika nycklar, två rader får inte innehålla dubbla värden eller kombinationer av värden för den specificerade kolumnen eller kolumnerna. En CREATE TABLE-sats kan ha flera UNIQUE-begränsningar, inklusive flera kolumner med en UNIQUE-begränsning i kolumnens definition och/eller flernivåers UNIQUE-begränsningar.

En CHECK-begränsning definierar ett uttryck som är evaluerat och måste vara sant för att dataraden ska läggas till eller uppdateras. CHECK-uttrycket måste evalueras som ett booleskt värde.

En COLLATE-sats i en kolumndefinition specificerar vilken textkollationsfunktion som används vid jämförelse av textposter i kolumnen. Som standard används sorteringsfunktionen BINARY. En definition av COLLATE-satsen och kollationsfunktioner finns i COLLATE.

Begränsningen DEFAULT specificerar ett standardvärde som ska användas vid en INSERT. Värdet kan vara NULL, en strängkonstant eller ett tal. Standardvärdet kan också vara ett av de särskilda nyckelorden CURRENT_TIME, CURRENT_DATE eller CURRENT_TIMESTAMP (inte skiftlägeskänsliga). Om värdet är NULL, en strängkonstant eller ett tal, infogas detta värde i kolumnen om en INSERT-sats inte specificerar ett värde för kolumnen. Om värdet är CURRENT_TIME, CURRENT_DATE eller CURRENT_TIMESTAMP infogas aktuellt UTC-datum och/eller klockslag i kolumnen. För CURRENT_TIME är formatet HH:MM:SS. För CURRENT_DATE är formatet YYYY-MM-DD. Formatet för CURRENT_TIMESTAMP är YYYY-MM-DD HH:MM:SS.

Om du anger en PRIMARY KEY skapas normalt ett UNIQUE-index på motsvarande kolumn eller kolumner. Om däremot PRIMARY KEY-begränsningen i en enkel kolumn har datatypen INTEGER (eller någon av dess synonymer så som int), används kolumnen av databasen som primärnyckelord för tabellen. Detta innebär att kolumnen endast kan innehålla unika heltalsvärden. (I många SQLite-implementeringar leder endast kolumntypen INTEGER till att kolumnen fungerar som intern primärnyckel, men i Adobe AIR anger synonymer för INTEGER så som int också detta beteende.)

Om tabellen inte har en INTEGER PRIMARY KEY-kolumn, kommer en heltalsnyckel att automatiskt genereras när raden infogas. Det går alltid att komma åt primärnyckeln för en rad med hjälp av specialnamnen ROWID, OID eller _ROWID_. Dessa namn kan användas oavsett om det finns en explicit deklarerad INTEGER PRIMARY KEY eller ett internt genererat värde. Om tabellen däremot har en explicit INTEGER PRIMARY KEY, är namnet på kolumnen i resultatdata det verkliga kolumnnamnet i stället för specialnamnet.

En INTEGER PRIMARY KEY-kolumn kan även innehålla nyckelordet AUTOINCREMENT. När AUTOINCREMENT-nyckelordet används genererar och infogar databasen automatiskt en heltalsnyckel i INTEGER PRIMARY KEY-kolumnen när en INSERT-programsats som inte specificerar ett värde för kolumnen utförs.

Det får endast finnas en PRIMARY KEY-begränsning i en CREATE TABLE-sats. Begränsningen kan antingen vara en del av kolumnens definition eller av en enkel PRIMARY KEY-begränsning. En primärnyckelkolumn är implicit NOT NULL.

Alternativet conflict-clause följt av flera begränsningar används för att ange en alternativ konfliktlösningsalgoritm som ska användas för den begränsningen. Standardvärdet är ABORT. Olika begränsningar inom samma tabell kan ha olika konfliktlösningsalgoritmer som standard. Om en INSERT- eller UPDATE-programsats specificerar en annan konfliktlösningsalgoritm, används den algoritmen på den plats som har angetts i programsatsen CREATE TABLE. Delen ON CONFLICT i ”Särskilda programsatser och klausuler” på sidan 1119 innehåller mer information.

Ytterligare begränsningar, som begränsningen FOREIGN KEY, resulterar inte i ett fel men körningen ignorerar dem.

Om nyckelordet TEMP eller TEMPORARY finns mellan CREATE och TABLE visas den tabell som skapades endast inom samma databasanslutning (SQLConnection-instans). Den tas automatiskt bort när databasanslutningen stängs. Index som skapas på en temporär tabell är också temporära. Temporära tabeller och index lagras i separata filer skilt från huvuddatabasfilerna.

Om det valfria prefixet database-name är specificerat, skapas tabellen i en namngiven databas (en databas som var ansluten till SQLConnection-instansen genom att anropa metoden attach() med det specifika databasnamnet). Det är syntaktiskt fel att specificera både ett database-name-prefix och TEMP-nyckelordet såvida inte prefixet database-name är temp. Om inget databasnamn specificeras och TEMP-nyckelordet inte finns, skapas tabellen i huvuddatabasen (den databas som anslöts till SQLConnection-instansen med metoden open() eller openAsync()).

Det finns inga godtyckliga gränser för antal kolumner eller begränsningar i en tabell. Det finns heller ingen begränsning av mängden data i en rad.

Formen CREATE TABLE AS definierar tabellen som ett resultat av en fråga. Namnen på tabellkolumnerna är namnen på kolumnerna i resultatet.

Om den valfria IF NOT EXISTS-satsen finns med, och om en annan tabell med samma namn redan finns, kommer databasen att ignorera kommandot CREATE TABLE.

En tabell kan tas bort med programsatsen DROP TABLE och begränsade ändringar kan göras med programsatsen ALTER TABLE.

ALTER TABLE

Kommandot ALTER TABLE används för att byta namn på eller lägga till en ny kolumn i en befintlig tabell. Det går inte att ta bort en kolumn från en tabell.

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

Syntaxen RENAME TO används för att byta namn på den tabell som identifieras av [database-name.] table-name till new-table-name. Detta kommando kan inte användas för att flytta en tabell mellan databaser, utan endast för att byta namn på en tabell inom samma databas.

Om tabellen som byter namn har utlösare eller index, kommer dessa fortfarande att vara kopplade till tabellen efter namnbytet. Om det däremot finns vydefinitioner eller programsatser som aktiveras av utlösare, kommer de inte automatiskt att använda det nya tabellnamnet. Om en omdöpt tabell har associerade vyer eller utlösare, måste du manuellt ta bort och skapa nya utlösare eller vyer med det nya tabellnamnet.

Syntaxen ADD [COLUMN] används för att lägga till en ny kolumn i en befintlig tabell. Den nya kolumnen läggs alltid till efter de befintliga kolumnerna. Satsen column-def kan anta alla former som är tillåtna i en CREATE TABLE-programsats, med följande begränsningar:

- Kolumnen får inte ha begränsningarna PRIMARY KEY eller UNIQUE.
- Kolumnen får inte ha standardvärdet för CURRENT_TIME, CURRENT_DATE eller CURRENT_TIMESTAMP.
- Om en NOT NULL-begränsning har angetts, måste kolumnen ha ett annat standardvärde än NULL.

Körningstiden för programsatsen ALTER TABLE påverkas inte av mängden data i tabellen.

DROP TABLE

Programsatsen DROP TABLE tar bort en tabell som har lagts till med CREATE TABLE. Tabellen med namnet table-name är den tabell som tas bort. Tabellen raderas fullständigt från databasen och diskfilen. Det går inte att återskapa den. Dessutom tas alla index som är associerade med tabellen bort.

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

Som standard minskar inte programsatsen DROP TABLE storleken på databasfilen. Tomt utrymme i databasen bevaras och används i efterföljande INSERT-åtgärder. Använd metoden SQLConnection.clean() för att frigöra utrymme i databasen. Om parametern autoClean är inställd på true när databasen skapas, kommer utrymme att frigöras automatiskt.

Den valfria IF EXISTS-satsen undertrycker det fel som normalt genereras om tabellen inte existerar.

CREATE INDEX

Kommandot CREATE INDEX består av nyckelorden CREATE INDEX, följt av namnet på nytt index, nyckelordet ON, namnet på föregående tabell som ska indexeras och en lista inom parentes med namn på kolumner i tabellen vars värden ska användas för indexnyckeln.

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

Varje kolumnnamn kan följas av nyckelorden ASC eller DESC för att indikera sorteringsordningen, men sorteringsordningen ignoreras av körningsbiblioteket. Sortering görs alltid i stigande ordning.

Satsen COLLATE efter varje kolumnnamn definierar den sorteringssekvens som används för textvärdet i den kolumnen. Standardkollationssekvensen är den kollationssekvens som är definierad för den kolumnen i programsatsen CREATE TABLE. Om ingen kollationssekvens är specificerad, används kollationssekvensen BINARY. En definition av COLLATE-satsen och kollationsfunktioner finns i COLLATE.

Det finns inga begränsningar för hur många index som kan kopplas till en enda tabell. Det finns inte heller några begränsningar för antalet kolumner i ett index.

DROP INDEX

Programsatsen DROP INDEX tar bort ett index som har lagts till med programsatsen CREATE INDEX. Det specificerade indexet raderas fullständigt från databasfilen. Det enda sättet att återskapa indexet är att köra motsvarande CREATE INDEX-kommando.

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

Som standard minskar inte programsatsen DROP INDEX storleken på databasfilen. Tomt utrymme i databasen bevaras och används i efterföljande INSERT-åtgärder. Använd metoden SQLConnection.clean() för att frigöra utrymme i databasen. Om parametern autoClean är inställd på true när databasen skapas, kommer utrymme att frigöras automatiskt.

CREATE VIEW

Kommandot CREATE VIEW tilldelar en fördefinierad SELECT-programsats ett namn. Det nya namnet kan sedan användas i en FROM-sats i en annan SELECT-programsats i stället för ett tabellnamn. Vyer används vanligtvis för att förenkla frågor genom att kombinera komplexa (och frekvent använda) datamängder i en struktur som kan användas i andra åtgärder.

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS
select-statement
```

Om nyckelordet TEMP eller TEMPORARY finns mellan CREATE och VIEW är vyn som skapas endast synlig för den SQLConnection-instans som öppnade databasen, och det tas bort automatiskt när databasen stängs.

Om [database-name] specificeras, skapas vyn i den namngivna databasen (den databas som var ansluten till SQLConnection-instansen med metoden attach()) med det specificerade name-argumentet. Det är syntaktiskt fel att specificera både [database-name] och TEMP om inte [database-name] är temp. Om inget databasnamn specificeras och TEMP-nyckelordet inte finns, skapas vyn i huvuddatabasen (den databas som anslöts till SQLConnection-instansen med metoden open() eller openAsync()).

Vyerna är skrivskyddade. Programsatserna DELETE, INSERT och UPDATE kan inte användas på en vy om inte minst en utlösare av motsvarande typ (INSTEAD OF DELETE, INSTEAD OF INSERT, INSTEAD OF UPDATE) har definierats. Mer information om hur du skapar en utlösare för en vy finns i CREATE TRIGGER.

En vy tas bort från databasen med programsatsen DROP VIEW.

DROP VIEW

Programsatsen DROP VIEW tar bort en vy som har skapats av en CREATE VIEW-programsats.

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

Uttrycket view-name är namnet på den vy som ska släppas. Den tas bort från databasen, men inga data i de underliggande tabellerna påverkas.

CREATE TRIGGER

Programsatsen CREATE TRIGGER används för att lägga till utlösare i databasschemat. En utlösare är en databasåtgärd (trigger-action) som automatiskt utförs när en specificerad databashändelse (database-event) inträffar.

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      [BEFORE | AFTER] database-event
                      ON table-name
                      trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      INSTEAD OF database-event
                      ON view-name
                      trigger-action
database-event  ::=  DELETE |
                      INSERT |
                      UPDATE |
                      UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                      BEGIN
                        trigger-step ;
                        [ trigger-step ; ]*
                      END
trigger-step    ::=  update-statement |
                      insert-statement |
                      delete-statement |
                      select-statement
column-list     ::=  column-name [, column-name]*
```

En utlösare aktiveras när en DELETE, INSERT eller UPDATE för en viss databastabell inträffar, eller när en UPDATE av en eller flera specificerade kolumner i en tabell sker. Utlösare är permanenta om inte nyckelorden TEMP eller TEMPORARY används. I detta fall tas utlösaren bort när SQLConnection-instansens huvuddatabasanslutning stängs. Om ingen tidsinformation finns angiven (BEFORE eller AFTER) används standardvärdet BEFORE.

Endast FOR EACH ROW-utlösare stöds, och texten FOR EACH ROW är valfri. Med en FOR EACH ROW-utlösare körs programsatserna trigger-step för varje databasrad som infogas, uppdateras eller raderas av den programsats som aktiverade utlösaren. Detta sker när WHEN-satsuttrycket evalueras till true.

Om en WHEN-sats anges, kommer de SQL-programsatser som har specificerats som utlösarsteg endast att utföras för rader där WHEN-satsen är sann. Om en WHEN-sats saknas, körs SQL-programsatserna för alla rader.

Inom utlösarens definition (trigger-action-satsen) är värdena före och efter ändringarna av tabellen tillgängliga med hjälp av de särskilda tabellnamnen OLD och NEW. Strukturen på tabellerna OLD och NEW matchar strukturen på tabellen för vilken utlösaren skapas. OLD-tabellen innehåller alla rader som har ändrats eller raderats av utlösaren, med den status som gällde innan utlösaren aktiverades. Tabellen NEW innehåller alla rader som har ändrats eller skapats av utlösaren, med den status som gäller efter att utlösarens programsatser körts. Både WHEN-satsen och trigger-step-programsatserna har åtkomst till värdena från den rad som infogas, raderas eller uppdateras med hjälp av formen NEW.column-name och OLD.column-name, där column-name är namnet på en kolumn från den tabell som utlösaren hör till. Tillgängligheten för tabellreferenserna OLD och NEW beror på typen av database-event som utlösaren hanterar:

- INSERT – NEW-referenser är giltiga
- UPDATE – NEW- och OLD-referenser är giltiga
- DELETE – OLD-referenser är giltiga

Den specificerade tiden (BEFORE, AFTER eller INSTEAD OF) bestämmer när trigger-step-programsatser utförs i förhållande till infogning, ändring eller borttagning av den associerade raden. En ON CONFLICT-sats kan specificeras som en UPDATE- eller INSERT-programsats i ett trigger-step. Om en ON CONFLICT-sats däremot är en specificerad del av programsatsen som styr utlösaren, används i stället konflikthanteringsprincipen.

En INSTEAD OF-utlösare kan också skapas för en vy. Om en eller flera INSTEAD OF INSERT-, INSTEAD OF DELETE- eller INSTEAD OF UPDATE-utlösare definieras för en vy är det inte ett syntaktiskt fel att utföra associerade programsatstyper (INSERT, DELETE eller UPDATE) på vyn. I det fallet utförs en INSERT, DELETE eller UPDATE på vyn, och den associerade utlösaren aktiveras. Eftersom utlösaren är en INSTEAD OF-utlösare kommer de underliggande tabellerna inte att ändras av programsatsen som aktiverar utlösaren. Utlösarna kan däremot användas för att utföra ändringar på de underliggande tabellerna.

Det finns en sak du bör tänka på när du skapar en utlösare i en tabell med en INTEGER PRIMARY KEY-kolumn. Om en BEFORE-utlösare ändrar INTEGER PRIMARY KEY-kolumnen för en rad som uppdateras av programsatsen som aktiverar utlösaren, uppdateras den inte. Du kan komma runt problemet genom att skapa tabellen med en PRIMARY KEY-kolumn istället för en INTEGER PRIMARY KEY-kolumn.

En utlösare kan tas bort med hjälp av programsatsen DROP TRIGGER. När en tabell eller vy tas bort, kommer alla utlösare som är associerade med tabellen eller vyn också att tas bort.

Funktionen RAISE()

Den särskilda SQL-funktionen RAISE() kan användas i en trigger-step-programsats i en utlösare. Den här funktionen har följande syntax:

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

Om en av de första tre formerna anropas vid en utlösaråtgärd, utförs den specificerade ON CONFLICT-processåtgärden (ABORT, FAIL eller ROLLBACK) och den aktuella programsatsen avslutas. ROLLBACK betraktas som ett körningsfel, vilket medför att den SQLStatement-instans vars execute()-metod utfördes genererar en (SQLErrorEvent.ERROR)-felhändelse. Det SQLError-objekt som finns i det skickade händelseobjektets felegenskap, har details-egenskapen inställd på det error-message som finns angiven i RAISE()-funktionen.

När RAISE(IGNORE) anropas, överges resten av den aktuella utlösaren, programsatsen som aktiverade utlösaren och alla efterföljande utlösare. Inga databasändringar rullas tillbaka. Om programsatsen som aktiverar utlösaren själv är en del av en utlösare, kommer detta utlösarprogram att återuppta åtgärden i början av nästa steg. Mer information om konfliktlösningsalgoritmer finns i avsnittet ON CONFLICT (konfliktalgoritmer).

DROP TRIGGER

Programsatsen DROP TRIGGER tar bort en utlösare som har skapats med CREATE TRIGGER.

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

Utlösaren raderas från databasen. Observera att utlösarna släpps automatiskt när deras associerade tabeller släpps.

Särskilda programsatser och klausuler

Det här avsnittet beskriver flera uttryck som är tillägg till SQL-standarden, och två språkelement som kan användas i många satser, kommentarer och uttryck.

COLLATE

Satsen COLLATE används i SELECT-, CREATE TABLE- och CREATE INDEX-programsatserna för att specificera den jämförelsealgoritm som ska användas vid en jämförelse eller sortering.

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

Standardkollationstypen för kolumner är BINARY. Om BINARY används med värden från lagringsklassen TEXT, jämförs de minnesbyte som representerar värdet, oavsett aktuell textkodning.

Kollationen NOCASE gäller endast värden i lagringsklassen TEXT. Om NOCASE används, är jämförelsen inte skiftlägeskänslig.

Ingen kollationssekvens används för lagringsklasser av typen NULL, BLOB, INTEGER eller REAL.

För att använda en annan kollationstyp än BINARY med en kolumn, måste COLLATE anges som en del av kolumndefinitionen i CREATE TABLE. När två TEXT-värden jämförs, används en kollationssekvens för att bestämma resultatet av jämförelsen enligt följande regler:

- För binära jämförelseoperatorer är standardkollationstypen för kolumnen det som bestämmer den kollationssekvens som används för jämförelsen, om någon av operanderna är en kolumn. Om båda operanderna är kolumner, bestämmer kollationstypen för den vänstra operanden vilken sekvens som används. Om ingen operand är en kolumn, används kollationssekvensen BINARY.
- Operatorn BETWEEN...AND är ekvivalent med användning av två uttryck med operatorerna >= och <=. Till exempel är uttrycket x BETWEEN y AND z ekvivalent med x >= y AND x <= z. Alltså följer operatorn BETWEEN...AND föregående regel för att bestämma kollationssekvensen.
- Operatorn IN fungerar som operatorn = för att bestämma den kollationssekvens som ska användas. Till exempel är den kollationssekvens som används för uttrycket x IN (y, z) en standardkollationstyp för x om x är en kolumn. Annars används kollationen BINARY.
- En ORDER BY-sats som är en del av ett SELECT-uttryck kan explicit tilldelas en kollationssekvens för att användas för sorteringsåtgärden. I detta fall används alltid den explicita kollationssekvensen. Om uttrycket som sorteras av ORDER BY är en kolumn, används i stället kolumnens standardkollationstyp för att bestämma sorteringsordningen. Om uttrycket inte är en kolumn, används kollationssekvensen BINARY.

EXPLAIN

Kommandomodifieraren EXPLAIN är ett tillägg som inte ingår i SQL-standarden.

```
sql-statement  ::=  EXPLAIN sql-statement
```

Om nyckelordet EXPLAIN står före andra SQL-programsatser körs inte själva kommandot. Däremot ger resultatet sekvensen av de virtuella maskininstruktioner som skulle ha använts för att utföra kommandot, om nyckelordet EXPLAIN inte hade funnits med. Funktionen EXPLAIN är en avancerad funktion som underlättar för utvecklare att ändra SQL-programsatser i ett försök att optimera prestanda eller felsöka en programsats som inte verkar fungera ordentligt.

ON CONFLICT (konfliktalgoritmer)

Satsen ON CONFLICT är inte ett separat SQL-kommando. Det är en sats som inte ingår i standarden och som kan förekomma i andra SQL-kommandon.

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

Den första formen av ON CONFLICT-satsen med nyckelorden ON CONFLICT används i en CREATE TABLE-programsats. För programsatsen INSERT eller UPDATE används den andra formen med ON CONFLICT som ersätts av OR för att få en mer naturlig syntax. Till exempel blir programsatsen INSERT OR IGNORE i stället för INSERT ON CONFLICT IGNORE. Även om nyckelorden är olika är båda satserna semantiskt ekvivalenta.

Satsen ON CONFLICT specificerar den algoritm som används för att lösa konflikter mellan begränsningar. De fem algoritmerna är ROLLBACK, ABORT, FAIL, IGNORE och REPLACE. Standardalgoritmen är ABORT. Här följer en förklaring av de fem konliktalgoritmerna:

ROLLBACK När en konflikt mellan olika begränsningar inträffar sker en ROLLBACK direkt och den aktuella transaktionen slutförs. Kommandot avbryts och SQLStatement-instansen genererar en error-händelse. Om ingen transaktion är aktiv (annat än implicita transaktioner som skapas för varje kommando) fungerar denna algoritm på samma sätt som ABORT.

ABORT När konflikt mellan olika begränsningar inträffar, tar kommandot tillbaka tidigare ändringar som det har utfört och SQLStatement-instansen genererar en error-händelse. Ingen ROLLBACK utförs, och ändringar från tidigare kommandon inom en transaktion bevaras. ABORT är standardbeteendet.

FAIL När en konflikt mellan olika begränsningar inträffar, avbryts kommandot och SQLStatement-instansen genererar en error-händelse. Ändringar i databasen som programsatsen gjorde innan konflikten inträffade, bevaras och tas inte tillbaka. Om en UPDATE-programsats till exempel påträffar en konflikt på rad 100, bevaras de ändringar som gjordes på de första 99 raderna. Däremot försvinner de ändringar som har gjorts på rad 100 och framåt.

IGNORE När en konflikt mellan olika begränsningar inträffar, ändras eller infogas inte den rad där konflikten inträffade. Förutom att den här raden ignoreras, fortsätter kommandot normalt. Andra rader före och efter raden som innehåller konfliken infogas eller uppdateras normalt. Inget fel returneras.

REPLACE När en UNIQUE-konflikt mellan olika begränsningar inträffar, tas föregående rader som orsakade konflikten, innan infogning eller uppdatering av aktuell rad, bort. Följaktligen görs alltid infogning eller uppdatering och sedan fortsätter kommandot normalt. Inget fel returneras. Om en NOT NULL-konflikt inträffar, ersätts NULL-värdet med standardvärdet för den kolumnen. Om kolumnen inte har något standardvärde används ABORT-algoritmen. Om en CHECK-överträdelse inträffar, används algoritmen IGNORE. När den här konfliktlösningsstrategin tar bort rader för att tillgodose en begränsning, aktiveras inte raderingsutlösarna för dessa rader.

Den algoritm som anges i OR-satsen i en INSERT- eller UPDATE-programsats åsidosätter de algoritmer som anges med CREATE TABLE. Om ingen algoritm är specificerad i CREATE TABLE eller i den INSERT eller UPDATE som körs, används ABORT.

REINDEX

Kommandot REINDEX används för att ta bort och regenerera ett eller flera index. Det här kommandot kan användas när definitionen av en kollationssekvens har ändrats.

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

Med den första formen återskapas alla index i relevanta databaser som använder den namngivna kollationssekvensen. I den andra formen, när en table-name specificeras, återskapas alla index som är associerade med tabellen. Om ett index-name är givet, återskapas och raderas endast specificerat index.

COMMENTS

Kommentarer är inte SQL-kommandon, men de kan förekomma i SQL-databasfrågor. De behandlas som blanktecken i körningsbiblioteket. De kan börja var som helst där det kan finnas ett blanktecken, inklusive inuti uttryck som sträcker sig över flera rader.

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

En enkelradskommentar anges med två tankstreck. En enkelradskommentar sträcker sig endast till slutet på den aktuella raden.

Blockkommentarer kan sträcka sig över flera rader eller vara inbäddade i en enkelrad. Om det inte finns en avslutande avgränsare, fortsätter blockkommentarer till slutet av inmatningen. Detta hanteras inte som ett syntaktiskt fel. En ny SQL-programsats kan börja på en rad efter att en blockkommentar slutar. Blockkommentarer kan bäddas in var som helst där det kan finnas ett blanktecken, inklusive inuti uttryck och i mitten av en annan SQL-programsats. Blockkommentarer kan inte kapslas i varandra. Enkelradskommentarer inuti en blockkommentar ignoreras.

EXPRESSIONS

Uttryck är underkommandon i andra SQL-block. Här nedan beskrivs giltig syntax för uttryck inom en SQL-programsats:

```
expr            ::=  expr binary-op expr |
                     expr [NOT] like-op expr [ESCAPE expr] |
                     unary-op expr |
                     ( expr ) |
                     column-name |
                     table-name.column-name |
                     database-name.table-name.column-name |
                     literal-value |
                     parameter |
                     function-name( expr-list | * ) |
                     expr ISNULL |
                     expr NOTNULL |
                     expr [NOT] BETWEEN expr AND expr |
                     expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name |
                     [EXISTS] ( select-statement ) |
                     CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                     CAST ( expr AS type ) |
                     expr COLLATE collation-name
like-op         ::=  LIKE | GLOB
binary-op       ::=  see Operators
unary-op        ::=  see Operators
parameter       ::=  :param-name | @param-name | ?
value-list      ::=  literal-value [, literal-value]*
literal-value   ::=  literal-string | literal-number | literal-boolean | literal-blob |
literal-null
literal-string  ::=  'string value'
literal-number  ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

Ett uttryck är en kombination av värden och operatorer som kan evalueras till ett enda värde. Uttryck kan delas in i två typer beroende på om de blir ett booleskt (sant eller falskt) värde eller om de blir ett icke booleskt värde.

I många situationer måste uttrycket kunna evalueras till ett booleskt värde, till exempel i en WHERE- eller HAVING-sats, i ON-uttrycket i en JOIN-sats och i ett CHECK-uttryck. Följande typer av uttryck uppfyller detta krav:

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()
- LIKE
- GLOB
- Vissa funktioner
- Vissa operatorer (specifika jämförelseoperatorer)

Literala värden

Ett literalt numeriskt värde skrivs som en heltalssiffra eller ett flyttal. Exponentformer stöds. Tecknet . (punkt) används alltid som decimaltecken.

En strängliteral indikeras genom att strängen står inom enkla citattecken ('). Om du vill använda ett enkelt citattecken i en sträng, sätter du två enkla citattecken i rad enligt följande:''.

Ett booleskt literalt värde indikeras av true eller false. Literala booleska värden används med kolumndatatypen Boolean.

En BLOB-literal är en strängliteral som innehåller hexadecimala data och föregås av tecknet x eller X, såsom i X'53514697465'.

Ett literalt värde kan även vara token NULL.

Kolumnnamn

Ett kolumnnamn kan vara något av namnen som definierades i programsatsen CREATE TABLE eller i en av följande särskilda identifierare: ROWID, OID eller _ROWID_. Dessa identifierare beskriver den unika heltalsslumpnyckeln (radnyckeln) som är kopplad till varje rad i varje tabell. De särskilda identifierarna refererar enbart till radnyckeln om programsatsen CREATE TABLE inte definierar en befintlig kolumn med samma namn. Radnycklar fungerar som skrivskyddade kolumner. En radnyckel kan användas var som helst där en vanlig kolumn kan användas, förutom att du inte kan ändra värdet för radnyckeln med UPDATE eller INSERT. SELECT * FROM "tabell" inkluderar inte radnyckeln i resultatet.

SELECT

Programsatsen SELECT kan förekomma i ett uttryck som antingen den högra operanden för operatorn IN eller som ett skalärvärde (ett entydigt resultatvärde), eller som en operand för operatorn EXISTS. När den används som ett skalärvärde eller som operand för en IN-operator, kan SELECT endast ha en enda kolumn i resultatet. En sammansatt SELECT-programsats (kopplad med nyckelord som UNION eller EXCEPT) är tillåten. Med EXISTS-operatorn, ignoreras kolumner i resultatet för SELECT, och uttrycket returnerar TRUE om det finns en eller flera rader eller FALSE om resultatmängden är tom. Om ingen term i SELECT-uttrycket refererar till värdet i frågan, kommer uttrycket att evalueras en gång innan annan bearbetning utförs och resultatet kan användas flera gånger vid behov. Om SELECT-uttrycket innehåller variabler från den yttre frågan (kallas även korrelerad underfråga) kommer SELECT att evalueras varje gång det behövs.

När SELECT är den högra operanden för operatorn IN, returnerar operatorn IN TRUE om resultatet från den vänstra operanden är detsamma som något av värdena i SELECT-programsatsens resultat. Operatorn IN kan föregås av nyckelordet NOT för att invertera testet.

När en SELECT förekommer i ett uttryck utan att vara den högra operanden för operatorn IN, blir den första raden i resultatet av SELECT det värde som används i uttrycket. Om SELECT ger mer än en resultatrad, kommer alla rader efter den första att ignoreras. Om SELECT inte ger några rader är värdet för SELECT lika med NULL.

CAST

Uttrycket CAST konverterar värdets datatyp till en given datatyp. Datatypen kan vara vilket icke-tomt namn som helst som är giltigt för typen i en kolumndefinition i en CREATE TABLE-programsats. Se Datatyper som stöds.

Övriga uttryckselement

Här beskrivs andra SQL-element som kan användas i uttryck:

- Inbyggda funktioner: Sammanställningsfunktioner, Skalärfunktioner och Formateringsfunktioner för datum och tid
- Operatorer
- Parametrar

Inbyggda funktioner

De inbyggda funktionerna delas upp i tre huvudkategorier:

- Sammanställningsfunktioner
- Skalärfunktioner
- Funktioner för datum och tid

Förutom dessa funktioner finns den särskilda funktionen RAISE() som används för att ge ett felmeddelande i utförandet av en utlösare. Den här funktionen kan endast användas i definitionen av programsatsen CREATE TRIGGER. Mer information om funktionen RAISE() finns i CREATE TRIGGER > RAISE().

Funktionsnamnen är som alla nyckelord i SQL inte skiftlägeskänsliga.

Sammanställningsfunktioner

Sammanställningsfunktioner utför åtgärder på värden från flera rader. Dessa funktioner används primärt i SELECT-programsatser i kombination med GROUP BY.

AVG(X)	Returnerar medelvärdet av alla icke-NULL X inom en grupp. Sträng- och BLOB-värden som inte ser ut som siffror tolkas som 0. Resultatet av AVG() är alltid ett flyttalsvärde, även om alla ingående värden är heltal.
COUNT(X) COUNT(*)	Den första formen returnerar antalet X som inte är NULL i en grupp. Den andra formen (med argumentet *) returnerar det totala antalet rader i en grupp.
MAX(X)	Returnerar det maximala värdet av alla värden i gruppen. Den vanliga sorteringsordningen används för att bestämma maximum.
MIN(X)	Returnerar det minsta icke-NULL-värdet av alla värden i gruppen. Den vanliga sorteringsordningen används för att bestämma det minsta värdet. Om alla värden i gruppen är NULL, returneras NULL.
SUM(X) TOTAL(X)	Returnerar den numeriska summan av alla icke-NULL-värden i gruppen. Om alla värden är NULL returnerar SUM() NULL och TOTAL() returnerar 0.0. Resultatet av TOTAL() är alltid ett flyttal. Resultatet av SUM() är ett heltalsvärde om alla ingående värden som är icke-NULL är heltal. Om ett ingående värde till SUM() inte är ett heltal och inte NULL, returnerar SUM() ett flyttal. Det här värdet kan vara en approximation av den verkliga summan.

I alla de föregående sammanställningsfunktionerna som tar ett argument, kan det argumentet föregås av nyckelordet DISTINCT. I så fall dupliceras element som är filtrerade innan de skickas till sammanställningsfunktionen. Funktionen anropar till exempel COUNT(DISTINCT x) som returnerar antal distinkta värden i kolumn X istället för totalt antal icke-NULL-värden i kolumn x.

Skalärfunktioner

Skalärfunktioner arbetar med värden från en rad i taget.

ABS(X)	Returnerar det absoluta värdet av argumentet X.
COALESCE(X, Y, ...)	Returnerar en kopia av det första icke-NULL-argumentet. Om alla argument är NULL, returneras NULL. Det måste finnas minst två argument.
GLOB(X, Y)	Den här funktionen används för att implementera syntaxen X GLOB Y.
IFNULL(X, Y)	Returnerar en kopia av det första icke-NULL-argumentet. Om bägge argumenten är NULL, returneras NULL. Den här funktionen fungerar på samma sätt som COALESCE().
HEX(X)	Argumentet tolkas som ett värde med lagringstypen BLOB. Resultatet är en hexadecimal återgivning av värdet.
LAST_INSERT_ROWID( )	Returnerar radidentifieraren (genererad primärnyckel) för den sista raden som har infogats i databasen via aktuell SQLConnection-instans. Det här värdet är samma som det värde som returneras av egenskapen SQLConnection.lastInsertRowID.
LENGTH(X)	Returnerar stränglängden för X i antal tecken.
LIKE(X, Y [, Z])	Den här funktionen används för att implementera X LIKE Y [ESCAPE Z] för SQL. Om den valfria ESCAPE-satsen finns, anropas funktionen med tre argument. I annat fall anropas den endast med två argument.
LOWER(X)	Returnerar en kopia av strängen X med alla tecken konverterade till gemener.
LTRIM(X) LTRIM(X, Y)	Returnerar en sträng som har formats genom borttagning av blanktecken till vänster om X. Om ett Y-argument har specificerats tar funktionen bort tecknen i Y från början av X.
MAX(X, Y, ...)	Returnerar argumentet med det högsta värdet. Argumentet kan vara strängar eller tal. Det högsta värdet bestäms av den definierade sorteringsordningen. Observera att MAX() är en enkel funktion när den har två eller flera argument, men att den är en sammanställningsfunktion när den har ett argument.
MIN(X, Y, ...)	Returnerar det argument som har det minsta värdet. Argumentet kan vara strängar eller tal. Det minsta värdet bestäms av den definierade sorteringsordningen. Observera att MIN() är en enkel funktion när den har två eller flera argument, men att den är en sammanställningsfunktion när den har ett argument.
NULLIF(X, Y)	Returnerar det första argumentet om argumenten inte är lika, annars returneras NULL.
QUOTE(X)	Den här funktionen returnerar en sträng som är värdet av dess argument i ett format som kan användas för infogning i en annan SQL-programsats. Strängarna är omgivna av enkla citattecken men föregås av inre citattecken vid behov. BLOB-lagringsklasser kodas som hexadecimala literaler. Funktionen är användbar i en utlösare som implementerar funktionen ångra/gör om.
RANDOM(*)	Returnerar ett pseudoslumpmässigt heltal mellan -9223372036854775808 och 9223372036854775807 Det här slumpmässiga värdet är inte kryptografiskt starkt.
RANDOMBLOB(N)	Returnerar en BLOB med N antal pseudoslumpmässiga byte. N ska vara ett positivt heltal. Det här slumpmässiga värdet är inte kryptografiskt starkt. Om värdet av N är negativt, returneras en enda byte.
ROUND(X) ROUND(X, Y)	Rundar av talet X till Y siffror till höger om decimaltecknet. Om argumentet Y utelämnas används 0.
RTRIM(X) RTRIM(X, Y)	Returnerar en sträng som har bildats genom att blanktecken i slutet av X tas bort. Om ett Y-argument har specificerats tar funktionen bort tecknen i Y från höger sida av X.
SUBSTR(X, Y, Z)	Returnerar en delsträng av indatasträngen X som börjar med det Y:e tecknet som är Z tecken långt. Tecknet längst till vänster i X är indexpositionen 1. Om Y är negativt, hittas första tecknet i delsträngen genom att funktionen räknar från höger och inte från vänster.
TRIM(X) TRIM(X, Y)	Returnerar en sträng som har bildats genom att blanktecken i slutet av X tas bort. Om ett Y-argument har specificerats tar funktionen bort tecknen i Y från höger sida av X.
TYPEOF(X)	Returnerar typen för uttrycket X. Möjliga returvärden är 'null', 'integer', 'real', 'text' och 'blob'. Mer information om datatyper finns i Datatyper som stöds.
UPPER(X)	Returnerar en kopia av indatasträngen X konverterad till versaler.
ZEROBLOB(N)	Returnerar en BLOB som innehåller N antal nollbyte (0x00).

Formateringsfunktioner för datum och tid

Formateringsfunktioner för datum och tid är en grupp skalärfunktioner som används för att skapa formaterade datum- och tidsvärden. Observera att dessa funktioner bearbetar och returnerar strängar och talvärden. Dessa funktioner är inte tänkta att användas med datatypen DATE. Om du använder dessa funktioner på data i en kolumn vars datatyp är DATE, kommer de inte att ge förväntat resultat.

DATE(T, ...)	Funktionen DATE() returnerar en sträng som innehåller data i det här formatet: YYYY-MM-DD. Den första parametern (T) specificerar en tidssträng i det format som beskrivs i Tidsformat. Ett valfritt antal modifierare kan anges efter tidssträngen. Modifierare beskrivs i Modifierare.
TIME(T, ...)	Funktionen TIME() returnerar en sträng med tiden i formatet HH:MM:SS. Den första parametern (T) specificerar en tidssträng i det format som beskrivs i Tidsformat. Ett valfritt antal modifierare kan anges efter tidssträngen. Modifierare beskrivs i Modifierare.
DATETIME(T, ...)	Funktionen DATETIME() returnerar en sträng med datum och tid i formatet YYYY-MM-DD HH:MM:SS. Den första parametern (T) specificerar en tidssträng i det format som beskrivs i Tidsformat. Ett valfritt antal modifierare kan anges efter tidssträngen. Modifierare beskrivs i Modifierare.
JULIANDAY(T, ...)	Funktionen JULIANDAY() returnerar en siffra som visar antalet dagar mellan 12.00 GMT, 24 november, 4 714 f.Kr. och det givna datumet. Den första parametern (T) specificerar en tidssträng i det format som beskrivs i Tidsformat. Ett valfritt antal modifierare kan anges efter tidssträngen. Modifierare beskrivs i Modifierare.
STRFTIME(F, T, ...)	STRFTIME()-funktionen returnerar data i det format som anges av formatsträngen som specificeras med det första argumentet F. Formatsträngen stöder följande ersättningar: %d – dag i månaden %f - sekunder i bråkdelar SS.SSS %H – timme 00-24 %j – dag på året 001-366 %J – dagens julianska värde %m – månad 01-12 %M – minut 00-59 %s – sekunder sedan 1970-01-01 %S – sekunder 00-59 %w – veckodag 0-6 (söndag = 0) %W – vecka på året 00-53 %Y – år 0000-9999 %% – % Den andra parametern (T) specificerar en tidssträng i det format som beskrivs i Tidsformat. Ett valfritt antal modifierare kan anges efter tidssträngen. Modifierare beskrivs i Modifierare.

Tidsformat

En tidssträng kan ha följande format:

YYYY-MM-DD	2007-06-15
YYYY-MM-DD HH:MM	2007-06-15 07:30
YYYY-MM-DD HH:MM:SS	2007-06-15 07:30:59
YYYY-MM-DD HH:MM:SS.SSS	2007-06-15 07:30:59.152
YYYY-MM-DDTHH:MM	2007-06-15T07:30
YYYY-MM-DDTHH:MM:SS	2007-06-15T07:30:59
YYYY-MM-DDTHH:MM:SS.SSS	2007-06-15T07:30:59.152
HH:MM	07:30 (datum är 2000-01-01)
HH:MM:SS	07:30:59 (datum är 2000-01-01)
HH:MM:SS.SSS	07:30:59:152 (datum är 2000-01-01)
now	Aktuellt datum och tid i Universal Coordinated Time.
DDDD.DDDD	Den julianska dagen anges med ett flyttal

Tecknet T i dessa format är det literala tecknet "T" som avgränsar datum och tid. Format som endast inkluderar en tid får datumet 2001-01-01.

Modifierare

Tidssträngen kan följas av noll eller flera modifierare som ändrar datum eller ändrar tolkningen av datumet. Tillgängliga modifierare är:

NNN days	Antal dagar att lägga till tiden.
NNN hours	Antal timmar att lägga till tiden.
NNN minutes	Antal minuter att lägga till tiden.
NNN.NNNN seconds	Antal sekunder och millisekunder att lägga till tiden.
NNN months	Antal månader att lägga till tiden.
NNN years	Antal år att lägga till tiden.
start of month	Skifta tid bakåt till början av månaden.
start of year	Skifta tid bakåt till början av året.
start of day	Skifta tid bakåt till början av dagen.
weekday N	Flyttar tiden till specificerad veckodag. (0 = söndag, 1 = måndag o.s.v.)
localtime	Konverterar datum till lokal tid.
utc	Konverterar datum till Universal Coordinated Time.

Operatorer

SQL stöder en stor mängd operatorer, både vanliga operatorer som finns i de flesta programspråk och operatorer som är unika för SQL.

Vanliga operatorer

Följande binära operatorer tillåts i ett SQL-block och visas sorterade från högsta till lägsta prioritetsordning:

```
*     /     %
+     -
<<  >>  &  |
<  >=    >  >=
=     ==    !=    <>  IN
AND
OR
```

Unära prefixoperatorer som stöds är:

```
 !     ~     NOT
```

Operatorn COLLATE motsvarar en unär postfix-operator. Operatorn COLLATE har prioritet framför alla andra operatorer. Den binder alltid hårdare än andra unära prefix-operatorer eller binära operatorer.

Observera att det finns två varianter av likhetsoperatorer. Likheten kan vara antingen = eller ==. De operatorer som uttrycker skillnad kan vara antingen != eller <>.

Operatorn || ger strängsammanfogning —. Den sammanfogar de två operandsträngarna.

Operatorn % ger resten vid division mellan vänster och höger operand (modulo).

Resultatet av binära operatorer är ett numeriskt värde förutom för sammanfogningsoperatorn || som ger ett strängresultat.

SQL-operatorer

LIKE

Operatorn LIKE gör en mönstermatchningsjämförelse.

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

Operanden till höger om LIKE innehåller mönstret och operanden till vänster innehåller strängen som ska matchas mot mönstret. Ett procenttecken (%) i mönstret fungerar som jokertecken och det matchar sekvenser med noll eller flera tecken i strängen. Ett understreck (_) i mönstret matchar ett tecken i strängen. Andra tecken matchar sig själva eller motsvarande små/stora bokstäver, det vill säga matchningen är inte skiftlägeskänslig. (Obs! Databasmotorn hanterar bara skillnaden mellan gemener och versaler för 7-bitars latinska tecken. Därför är operatorn LIKE skiftlägeskänslig för 8-bitars iso8859-tecken och UTF-8-tecken. Till exempel är uttrycket 'a' LIKE 'A' TRUE, men 'æ' LIKE 'Æ' är FALSE). Skiftlägeskänsligheten för latinska tecken kan ändras med egenskapen SQLConnection.caseSensitiveLike.

Om den valfria satsen ESCAPE finns med, måste uttrycket som följer nyckelordet ESCAPE evalueras till en sträng som består av ett tecken. Det här tecknet kan användas i LIKE-mönstret för att matcha literala procent- eller understrecktecken. Escape-tecknet följt av en procentsymbol, ett understreck eller sig själv matchar en literal procentsymbol, understreck eller escape-tecken i strängen.

GLOB

Operatorn GLOB påminner om LIKE men använder Unix-syntaxen för jokertecken. Till skillnad från LIKE är operatorn GLOB skiftlägeskänslig.

IN

Operatorn IN beräknar om dess vänstra operand har samma värde som ett av värdena i den högra operanden (en mängd värden inom parentes).

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

Den högra operanden kan vara en uppsättning av kommaseparerade literala värden, eller så kan den vara ett resultat av en SELECT-programsats. Se även SELECT-programsatser i uttryck för en förklaring och begränsningar av hur du kan använda SELECT som högeroperand för operatorn IN.

BETWEEN...AND

Operatorn BETWEEN...AND är ekvivalent med användning av två uttryck med operatorerna >= och <=. Till exempel är uttrycket x BETWEEN y AND z lika med x >= y AND x <= z.

NOT

Operatorn NOT är negationsoperatorn i SQL. Operatorerna GLOB, LIKE och IN kan föregås av nyckelordet NOT för att invertera logiken (med andra ord kontrollera att värdet inte matchar det angivna mönstret).

Parametrar

En parameter anger en platshållare i uttrycket för ett literalt värde som fylls i vid körning genom att ett värde tilldelas till den associativa arrayen SQLStatement.parameters. Parametrar kan ha tre former:

?	Ett frågetecken indikerar en indexparameter. Parametrar tilldelas numeriska (nollbaserade) indexvärden enligt deras ordning i programsatsen.
:AAAA	Ett kolon följt av ett identifierarnamn är platshållare för en namngiven parameter med namnet AAAA. Namngivna parametrar är även numrerade enligt deras ordningsföljd i SQL-programsatsen. För att undvika förvirring är det bäst att undvika att blanda namngivna och numrerade parametrar.
@AAAA	Ett snabel-a motsvarar ett kolon.

SQL-funktioner som inte stöds

Här följer en lista på standardelement i SQL som inte stöds i Adobe AIR:

FOREIGN KEY-restriktioner FOREIGN KEY-restriktionerna tolkas men används inte.

Utlösare: FOR EACH STATEMENT-utlösare stöds inte (alla utlösare måste vara FOR EACH ROW). INSTEAD OF-utlösare stöds inte i tabeller (INSTEAD OF-utlösare tillåts endast i vyer). Rekursiva utlösare, utlösare som aktiverar sig själva, stöds inte.

ALTER TABLE Endast varianterna RENAME TABLE och ADD COLUMN för ALTER TABLE-kommandon stöds. Andra slags ALTER TABLE-åtgärder som DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT och så vidare, ignoreras.

Kapslade transaktioner Endast en aktiv transaktion är tillåten.

RIGHT och FULL OUTER JOIN RIGHT OUTER JOIN och FULL OUTER JOIN stöds inte.

Uppdaterbar VIEW En vy är skrivskyddad. Du kan inte köra en DELETE-, INSERT- eller UPDATE-programsats på en vy. En INSTEAD OF-utlösare som aktiveras som ett försök att utföra DELETE, INSERT eller UPDATE på en vy, stöds och kan användas för att uppdatera tabeller som stöds i utlösarens innehåll.

GRANT och REVOKE En databas är som andra filer på disken och den enda åtkomstbehörighet som är relevant är samma åtkomstbehörighet som finns i operativsystemet. Kommandona GRANT och REVOKE finns vanligtvis i klient/servermiljöer, RDBMS:er är inte implementerade.

Följande SQL-element och SQLite-funktioner stöds i vissa SQLite-implementeringar, men inte i Adobe AIR. Den mesta av denna funktionalitet finns tillgänglig genom metoder i klassen SQLConnection:

Transaktionsrelaterade SQL-element (BEGIN, END, COMMIT, ROLLBACK) Den här funktionaliteten finns tillgänglig genom transaktionsrelaterade metoder i klassen SQLConnection: SQLConnection.begin(), SQLConnection.commit() och SQLConnection.rollback().

ANALYZE Den här funktionaliteten finns tillgänglig genom metoden SQLConnection.analyze().

ATTACH Den här funktionaliteten finns tillgänglig genom metoden SQLConnection.attach().

COPY Den här programsatsen stöds inte.

CREATE VIRTUAL TABLE Den här programsatsen stöds inte.

DETACH Den här funktionaliteten finns tillgänglig genom metoden SQLConnection.detach().

PRAGMA Den här programsatsen stöds inte.

VACUUM Den här funktionaliteten finns tillgänglig genom metoden SQLConnection.compact().

Systemtabellåtkomst är inte möjlig Systemtabeller inklusive sqlite_master och andra tabeller med prefixet ”sqlite_” finns inte tillgängliga i SQL-programsatser. Körtidsbiblioteket inkluderar ett API-schema som ger ett objektorienterat sätt att komma åt schemadata. Mer information finns i metoden SQLConnection.loadSchema().

Funktioner för reguljära uttryck (MATCH() och REGEX()) Dessa funktioner finns inte tillgängliga för SQL-programsatser.

Följande funktionalitet skiljer sig mellan många SQLite-implementeringar och Adobe AIR:

Indexerade programsatsparametrar I många implementeringar är indexerade programsatsparametrar 1-baserade. I Adobe AIR är parametrarna nollbaserade (den första parametern ges index 0, den andra parametern ges index 1 o.s.v.).

INTEGER PRIMARY KEY-kolumndefinitioner I många implementeringar används bara kolumner som har definierats exakt som INTEGER PRIMARY KEY som primärnyckelkolumn för en tabell. I dessa implementeringar, med en annan datatyp som vanligtvis är synonym för INTEGER (t.ex. int), fungerar inte kolumnen som intern primärnyckel.

I Adobe AIR betraktas däremot datatypen int (och andra INTEGER-synonymer) som exakta motsvarigheter till INTEGER. Därför används en kolumn som definierats som int PRIMARY KEY som intern primärnyckel för en tabell. Mer information finns i avsnitten CREATE TABLE och Kolumntillhörighet.

Ytterligare SQL-funktioner

Följande kolumndatatyper stöds inte i SQLite men däremot i Adobe AIR. (Observera att dessa datatypsnamn, som alla nyckelord i SQL, inte är skiftlägeskänsliga):

Boolean motsvarar klassen Boolean.

Date motsvarar klassen Date.

int motsvarar klassen int (samma som kolumntillhörigheten INTEGER).

Number motsvarar klassen Number (samma som kolumntillhörigheten REAL).

Object motsvarar klassen Object eller underklasser som kan serialiseras och deserialiseras med AMF3. (Detta inkluderar de flesta klasser som innehåller egna klasser, men utesluter vissa klasser inklusive visningsobjekt och objekt som innehåller visningsobjekt som egenskaper.)

String motsvarar klassen String (samma som kolumntillhörigheten TEXT).

XML motsvarar ActionScript-klassen (E4X) XML.

XMLList motsvarar ActionScript-klassen (E4X) XMLList.

Följande literala värden stöds inte i SQLite, men däremot i Adobe AIR:

true används för att representera det literala booleska värdet true, så att det fungerar med BOOLEAN-kolumner.

false används för att representera det literala booleska värdet false, så att det fungerar med BOOLEAN-kolumner.

Datatyper som stöds

Till skillnad från de flesta SQL-databaser kräver eller tvingar inte Adobe AIR SQL-databasen att tabellkolumnvärden har en viss typ. I stället används i körtidsbiblioteket två koncept, lagringsklasser och kolumntillhörighet, för att styra datatyperna. Det här avsnittet beskriver lagringsklasser och kolumntillhörighet och hur datatypsskillnader kan lösas i olika situationer:

- ”Lagringsklasser” på sidan 1131
- ”Kolumntillhörighet” på sidan 1131
- ”Datatyper och jämförelseoperatorer” på sidan 1134
- ”Datatyper och matematiska operatorer” på sidan 1134
- ”Datatyper och sortering” på sidan 1134
- ”Datatyper och gruppering” på sidan 1134
- ”Datatyper och sammansatta SELECT-programsatser” på sidan 1135

Lagringsklasser

Lagringsklasser representerar befintliga datatyper som används för att lagra värden i en databas. Följande lagringsklasser används i databasen:

NULL Värdet är ett NULL-värde.

INTEGER Värdet är ett positivt eller negativt heltal.

REAL Värdet är ett reellt flyttal.

TEXT Värdet är en textsträng (begränsad till 256 MB).

BLOB Värdet är ett BLOB-objekt (Binary Large Object), dvs. binära rådata (begränsat till 256 MB).

Alla värden som skickas till databasen som literaler inbäddade i SQL-programsatser eller bundna värden med parametrar till en förberedd SQL-programsats är tilldelade en lagringsklass innan SQL-programsatsen körs.

Literaler som är en del av en SQL-programsats tilldelas lagringsklassen TEXT om de står inom enkla eller dubbla citattecken, INTEGER om literalen är specificerad som en siffra utan citattecken och utan decimaltecken eller exponent, REAL om literalen är en siffra utan citattecken med ett decimaltecken, eller exponent eller NULL, om värdet är NULL. Literaler med lagringsklassen BLOB specificeras med notationen X'ABCD'. Mer information finns i Literala värden i uttryck.

Värden som ges som parametrar med SQLStatement.parameters tilldelas den lagringsklass som närmast matchar en intern datatypsgräns. Till exempel är integer-värden bundna till lagringsklassen INTEGER, numeriska värden ges lagringsklassen REAL, strängvärden ges lagringsklassen TEXT och ByteArray-objekt ges lagringsklassen BLOB.

Kolumntillhörighet

En kolumns *tillhörighet* är den rekommenderade datatypen i den kolumnen. När ett värde lagras i en kolumn (via INSERT eller UPDATE), försöker körtidsbiblioteket att konvertera det värdet från den faktiska datatypen till den specificerade tillhörigheten. Om till exempel ett värde med datatypen DATE (en ActionScript- eller JavaScript-instans) infogas i en kolumn vars tillhörighet är TEXT, kommer datumvärdet att konverteras till strängrepresentation (motsvarande anrop av objektets toString()-metod) innan värdet lagras i databasen. Om värdet inte kan konverteras till den specificerade tillhörigheten inträffar ett fel och åtgärden utförs inte. När ett värde hämtas från databasen med SELECT, returneras den som en instans av den klass som motsvarar tillhörigheten oavsett om den konverterades från en annan datatyp när den lagrades.

Om en kolumn accepterar NULL-värden, kan ActionScript- eller JavaScript-värdet null användas som ett parametervärde för att lagra NULL i kolumnen. När ett värde med lagringsklassen NULL hämtas med SELECT, returneras det som ActionScript- eller JavaScript-värdet null, oavsett kolumnens tillhörighet. Om en kolumn accepterar NULL-värden, måste du alltid kontrollera de värden som hämtas från den kolumnen för att bestämma om de är null innan du försöker konvertera värdet till en typ som inte kan vara null (som ett numeriskt eller ett booleskt värde).

Varje kolumn i databasen tilldelas en av följande typtillhörigheter:

- TEXT (eller String)
- NUMERIC
- INTEGER (eller int)
- REAL (eller Number)
- Boolean
- Date

- XML
- XMLLIST
- Object
- NONE

TEXT (eller String)

En kolumn med tillhörigheterna TEXT eller String lagrar alla data med lagringsklasserna NULL, TEXT eller BLOB. Om numeriska data infogas i en kolumn med tillhörigheten TEXT, konverteras de till textform innan de lagras.

NUMERIC

En kolumn med tillhörigheten NUMERIC innehåller värden där lagringsklasserna NULL, REAL eller INTEGER används. När textdata infogas i en NUMERIC-kolumn, görs ett försök att konvertera värdet till ett heltal eller till ett reellt tal innan det lagras. Om konvertering genomförs, lagras värdet med hjälp av lagringsklasserna INTEGER eller REAL (till exempel konverteras värdet '10,05' till lagringsklassen REAL innan det lagras). Om konverteringen inte kan utföras uppstår ett fel. Inget försök görs att konvertera ett NULL-värde. Ett värde som hämtas från en NUMERIC-kolumn returneras som en instans av den mest specifika numeriska typen till vilken värdet passar. Om värdet är ett positivt heltal eller 0, returneras det alltså som en uint-instans. Om det är ett negativt heltal, returneras det som en int-instans. Om värdet har en flyttalskomponent (värdet är inte ett heltal), returneras det som en Number-instans.

INTEGER (eller int)

En kolumn som använder tillhörigheten INTEGER fungerar på samma sätt som en kolumn med NUMERIC-tillhörighet, fast med ett undantag. Om värdet som ska lagras är ett reellt värde (som en instans av Number) utan flyttalskomponent, eller om värdet är ett textvärde som kan konverteras till ett reellt värde utan flyttalskomponent, konverteras det till ett heltal och lagras med lagringsklassen INTEGER. Om ett försök görs att lagra ett reellt värde med en flyttalskomponent uppstår ett fel.

REAL (eller Number)

En kolumn med tillhörigheten REAL eller NUMBER fungerar på samma sätt som en kolumn med tillhörigheten NUMERIC, förutom att den tvingar heltalsvärden till flyttalsrepresentation. Ett värde i en REAL-kolumn returneras alltid från databasen som en Number-instans.

Boolean

En kolumn med tillhörigheten Boolean lagrar sanna eller falska värden. En Boolean-kolumn accepterar ett värde som är en ActionScript- eller JavaScript-instans av Boolean. Om koden försöker lagra ett strängvärde, betraktas ett strängvärde med en längd större än noll som sann och en tom sträng som falsk. Om koden försöker lagra numeriska data, lagras värden som inte är noll som sanna och 0 lagras som falskt. När ett booleskt värde hämtas med SELECT, returneras värdet som en Boolean-instans. NULL-skilda värden lagras med lagringsklassen INTEGER (0 för falskt och 1 för sant) och konverteras till Boolean-objekt när data hämtas.

Date

En kolumn med tillhörigheten Date lagrar datum och tid. En Date-kolumn är avsedd att acceptera värden som är ActionScript- eller JavaScript-instanser av Date. Om försök görs att lagra ett strängvärde i en Date-kolumn, försöker körtidsbiblioteket konvertera värdet till ett julianskt datum. Om konverteringen misslyckas uppstår ett fel. Om koden försöker att lagra ett värde av typen Number, int eller uint, görs inga försök att validera data och värdet betraktas som ett giltigt julianskt datumvärde. Ett Date-värde som hämtas med SELECT konverteras automatiskt till en Date-instans. Date-värden lagras som julianska datumvärden med lagringsklassen REAL, och sorterings- och jämförelseåtgärder fungerar som förväntat.

XML eller XMLList

En kolumn som använder tillhörigheten XML- eller XMLList lagrar XML-strukturer. Om koden försöker lagra data i en XML-kolumn med en SQLStatement-parameter, försöker körtidsbiblioteket att konvertera och validera värdet med ActionScript-funktionen XML() eller XMLList(). Ett fel uppstår om värdet inte kan konverteras till giltig XML. Om försöket att lagra data använder ett literalt SQL-textvärde (till exempel INSERT INTO (col1) VALUES ('Invalid XML (ingen avslutningstagg)')), undersöks eller valideras inte värdet, utan textvärdet antas att vara välformad XML. Om ett ogiltigt värde lagras, returneras det när det hämtas som ett tomt XML-objekt. XML- och XMLList-data lagras med lagringsklassen TEXT eller NULL.

Object

En kolumn med tillhörigheten Object lagrar ActionScript- eller JavaScript-objekt som är komplexa, inklusive objektklassinstanser och instanser av programsatser för underklasser som arrayer och anpassade klassinstanser. Object-kolumndata är serialiserade i AMF3-format och lagras med lagringsklassen BLOB. När ett värde hämtas, deserialiseras det från AMF3 och returneras som en instans av samma klass som när den lagrades. Observera att några ActionScript-klasser, speciellt visningsobjekt, inte kan deserialiseras som instanser av deras originaldatatyp. Innan du lagrar en egen klassinstans måste du registrera ett alias för klassen med metoden flash.net.registerClassAlias() (eller i Flex genom att lägga till [RemoteObject]-metadata i klassdeklarationerna). Dessutom måste du innan du hämtar data registrera samma alias för klassen. Data som inte kan deserialiseras ordentligt, antingen på grund av att klassen inte kan deserialiseras eller att klassalias saknas eller är felkopplade, returneras som ett anonymt objekt (en Object-klassinstans) med egenskaper och värden som motsvarar den lagrade originalinstansen.

NONE

En kolumn med tillhörigheten NONE föredrar inte en viss lagringsklass framför en annan. Det görs inga försök att konvertera data innan de infogas.

Bestämma tillhörighet

Typtillhörigheten för en kolumn bestäms av den deklarerade typen för kolumnen i programsatsen CREATE TABLE. När typen bestäms gäller följande regler (inte skiftlägeskänsliga):

- Om kolumnens datatyp innehåller några av strängarna CHAR, CLOB, STRI eller TEXT har kolumnen tillhörigheten TEXT/String. Observera att typen VARCHAR innehåller strängen CHAR och därför tilldelas tillhörigheten TEXT.
- Om kolumnens datatyp innehåller strängen BLOB, eller om ingen datatyp är specificerad, har kolumnen tillhörigheten NONE.
- Om datatypen för kolumnen innehåller strängen XMLL, har kolumnen tillhörigheten XMLList.
- Om datatypen är strängen XML, har kolumnen tillhörigheten XML.
- Om datatypen innehåller strängen OBJE, har kolumnen tillhörigheten Object.
- Om datatypen innehåller strängen BOOL, har kolumnen tillhörigheten Boolean.
- Om datatypen innehåller strängen DATE, har kolumnen tillhörigheten Date.
- Om datatypen innehåller strängen INT (inklusive UINT) ges den tillhörigheten INTEGER/int.
- Om kolumnens datatyp innehåller några av strängarna REAL, NUMB, FLOA eller DOUB får kolumnen tillhörigheten REAL/Number.
- I annat fall är tillhörigheten NUMERIC.
- Om en tabell skapas med CREATE TABLE t AS SELECT... har inga kolumner en specificerad datatyp och de får samtliga tillhörigheten NONE.

Datatyper och jämförelseoperatorer

Följande binära jämförelseoperatorer =, <, <=, >= och != stöds tillsammans med en åtgärd för att testa för mängdmedlemskap IN och den ternära jämförelseoperatorn BETWEEN. Mer information om dessa operatorer finns i Operatorer.

Resultatet av en jämförelse beror på lagringsklasserna för de två värden som jämförs. Följande regler gäller när två värden jämförs:

- Ett värde med lagringsklassen NULL betraktas som mindre än vilket annat värde som helst (inklusive ett annat värde med lagringsklassen NULL).
- Ett INTEGER- eller REAL-värde är mindre än värdena TEXT eller BLOB. Om en INTEGER eller REAL jämförs med en annan INTEGER eller REAL, utförs en numerisk jämförelse.
- Ett TEXT-värde är mindre än ett BLOB-värde. När två TEXT-värden jämförs, utförs en binär jämförelse.
- När två BLOB-värden jämförs, bestäms alltid resultatet med hjälp av en binär jämförelse.

Den ternära operatorn BETWEEN typkonverteras alltid som det motsvarande binära uttrycket. Till exempel konverteras a BETWEEN b AND c till a >= b AND a <= c, även om det innebär att olika tillhörigheter ges till a i varje jämförelse som kräver evaluering av uttrycket.

Uttryck av typen a IN (SELECT b ....) hanteras av de tre reglerna som har beskrivits tidigare för binära jämförelser, det vill säga på liknande sätt som a = b. Exempel: Om b är ett kolumnvärde och a är ett uttryck, tillämpas tillhörigheten för b på a innan jämförelsen utförs. Uttrycket a IN (x, y, z) konverteras till a = +x OR a = +y OR a = +z. Värdena till höger om operatorn IN (värdena x, y och z i det här exemplet) betraktas som uttryck, även om de råkar vara kolumnvärden. Om värdet till vänster om operatorn IN är en kolumn, används kolumnens tillhörighet. Om värdet är ett uttryck sker ingen konvertering.

Hur jämförelser utförs kan även påverkas av en COLLATE-sats. Mer information finns i COLLATE.

Datatyper och matematiska operatorer

För varje matematisk operator som stöds, *, /, %, + och -, tillämpas en numerisk tillhörighet på varje operand innan uttrycket evalueras. Om en operand inte kan konverteras till lagringsklassen NUMERIC evalueras uttrycket till NULL.

När sammanfogningsoperatorn || används, konverteras varje operand till lagringsklassen TEXT innan uttrycket evalueras. Om en operand inte kan konverteras till lagringsklassen TEXT, blir resultatet av uttrycket NULL. Det kan i två situationer förekomma svårigheter konvertera värdet, om värdet av operanden är NULL, eller om en BLOB innehåller en annan lagringsklass än TEXT.

Datatyper och sortering

När värden sorteras med en ORDER BY-sats, kommer värden med lagringsklassen NULL först. Dessa följs av INTEGER- och REAL-värden i numerisk ordning, följt av TEXT-värden i binär ordning eller baserat på den specificerade kollationen (BINARY eller NOCASE). Slutligen kommer BLOB-värden i binär ordning. Ingen konvertering av lagringsklasser sker före sortering.

Datatyper och gruppering

Vid gruppering av värden med GROUP BY betraktas värden med skilda lagringsklasser som olika. Ett undantag är INTEGER- och REAL-värden som betraktas som lika om de är numeriskt lika. Inga tillhörigheter tillämpas på värden som ett resultat av en GROUP BY-sats.

Datatyper och sammansatta SELECT-programsatser

De sammansatta SELECT-operatorerna UNION, INTERSECT och EXCEPT används för att utföra implicita jämförelser mellan värden. Innan dessa jämförelser utförs måste varje värde ges en tillhörighet. Samma tillhörighet, om någon, tillämpas på alla värden som eventuellt returneras i en enda kolumn i den sammansatta SELECT-resultatmängden. Tillhörigheten som tillämpas är tillhörigheten för den kolumn som returneras av den första komponenten i SELECT-programsatsen som har ett kolumnvärde (och inte något annat uttryck) i det läget. Om det för en given sammansatt SELECT-kolumn inte returneras någon komponent av en ingående SELECT-programsats, tillämpas ingen tillhörighet på värden från den kolumnen innan de jämförs.

Kapitel 67: SQL-felmeddelanden, ID:n och argument

Klassen SQLError representerar olika fel som kan inträffa vid arbete med en lokal SQL-databas i Adobe AIR. För varje fel har SQLError-instansen egenskapen `details` som innehåller ett felmeddelande på svenska. Dessutom har varje felmeddelande en motsvarande unik identifierare som är tillgänglig i SQLError-objektets `detailID`-egenskap Med egenskapen `detailID` kan ett program identifiera det specifika details-felmeddelandet. Programmet kan ge alternativ text för slutanvändaren på hans/hennes språk. Argumentvärdena i `detailArguments`-arrayen kan ersättas på lämplig position i felmeddelandesträngen. Detta är användbart för program som vill visa `details`-egenskapens felmeddelande för detta fel direkt för slutanvändare på ett specifikt språk.

I följande tabell visas en lista på `detailID`-värden och den motsvarande svenska felmeddelandetexten. Platshållartext i meddelandena visar var `detailArguments`-värden ersätts vid körningen. Denna lista kan användas som en källa för att lokalisera de felmeddelanden som ges vid SQL-databasåtgärder.

SQLError detailID	Svenskt felmeddelande och parametrar
1001	Anslutningen stängdes.
1102	Databasen måste vara öppen när denna åtgärd genomförs.
1003	%s [,|och %s] parameternamn som hittades i parametrarnas egenskaper fanns inte i det angivna SQL-uttrycket.
1004	Felmatchning i parameterräkning. Hittade %d i angivet SQL-uttryck och %d-värden som angavs i parameterns egenskap. Förväntade värden för %s [,|och %s].
1005	Automatisk återvinning ska inte vara aktiverad.
1006	Det gick inte att ställa in pageSize-värdet.
1007	Schemaobjektet med namnet "%"s av typen "%s" i databasen "%s" hittades inte.
1008	Schemaobjektet med namnet "%s" i databasen "%s" hittades inte.
1009	Schemaobjektet med typen "%s" i databasen "%s" hittades inte.
1010	Inga schemaobjekt hittades i databasen "%s".
2001	Tolkningsspill i stacken.
2002	För många argument i funktionen "%s"
2003	Nära "%s": syntaxfel
2004	Det finns redan en annan tabell eller ett annat index med det här namnet: "%s"
2005	PRAGMA stöds inte i SQL.
2006	Ej skrivbar katalog.
2007	Okänd join-typ eller också stöds den inte: "%s %s %s"
2008	RIGHT och FULL OUTER JOIN stöds för närvarande inte.
2009	En NATURAL-join får inte ha en ON- eller USING-sats.
2010	Det går inte att ha både ON- och USING-satser i samma join.
2011	Det går inte att använda join med kolumnen "%s" - kolumnen finns inte i båda tabellerna.
2012	Endast ett resultat är tillåtet för SELECT som är del av ett uttryck.
2013	Ingen sådan tabell: "[%s.]%s"
2014	Ingen tabell har angivits.
2015	För många kolumner i resultatuppsättningen|för många kolumner i "%s".
2016	"%s" ORDER|GROUP BY-termen ligger utanför intervallet - den ska ligga mellan 1 och %d.
2017	För många termer i ORDER BY-satsen.
2018	%s ORDER BY-termen ligger utanför intervallet - den ska ligga mellan 1 och %d.
2019	%r ORDER BY-termen matchar ingen kolumn i resultatmängden.
2020	ORDER BY-satsen ska komma efter "%s", inte före.
2021	LIMIT-satsen ska komma efter "%s", inte före.
2022	SELECT till vänster och höger om "%s" har inte samma antal resultatkolumner.
2023	En GROUP BY-sats krävs innan HAVING.

2024	Sammanställningsfunktioner är inte tillåtna i GROUP BY-satsen.
2025	DISTINCT i sammanställningen måste följas av ett uttryck.
2026	För många termer i sammansatt SELECT.
2027	För många termer i ORDER|GROUP BY-satsen.
2028	En temporär utlösare får inte ha ett kvalificerat namn.
2030	Utlösaren "%s" finns redan.
2032	Det går inte att skapa BEFORE|AFTER-utlösare i vyn: '%s'.
2033	Det går inte att skapa INSTEAD OF-utlösare i tabellen: "%s".
2034	Ingen sådan utlösare: "%s"
2035	Rekursiva utlösare stöds inte ("%s").
2036	Ingen sådan kolumn: %s[.%s[.%s]]
2037	VACUUM tillåts inte från SQL.
2043	Tabellen "%s": indexeringsfunktionen returnerade en ogiltig plan.
2044	Max %d tabeller i en join.
2046	Det går inte att lägga till en PRIMARY KEY-kolumn.
2047	Det går inte att lägga till en UNIQUE-kolumn.
2048	Det går inte att lägga till en NOT NULL-kolumn med standardvärde NULL.
2049	Det går inte att lägga till en kolumn med en icke-konstant standard.
2050	Det går inte att lägga till en kolumn i en vy.
2051	ANALYZE tillåts inte i SQL.
2052	Ogiltigt namn: "%s"
2053	ATTACH tillåts inte från SQL.
2054	%s "%s" kan inte referera till objekt i databasen "%s".
2055	Åtkomst till "[%s.]%s.%s" är inte tillåten.
2056	Inte behörig.
2058	Ingen sådan vy: "[%s.]%s"
2060	Ett temporärt tabellnamn måste vara okvalificerat.
2061	Tabellen "%s" finns redan.
2062	Det finns redan ett index med namnet: "%s"
2064	Dubbla kolumnnamn: "%s"
2065	Tabellen "%s" har mer än en primärnyckel.
2066	AUTOINCREMENT tillåts bara på en INTEGER PRIMARY KEY.
2067	Ingen sådan kollationssekvens: "%s"
2068	Parametrar är inte tillåtna i vyer.
2069	Vyn "%s" är cirkeldefinierad.
2070	Tabellen "%s" får inte tas bort.
2071	Använd DROP VIEW för att ta bort vyn "%s".
2072	Använd DROP TABLE för att ta bort tabellen "%s".
2073	Foreign-nyckeln på "%s" ska endast referera till en kolumn i tabellen "%s".
2074	Antal kolumner i foreign-nyckeln matchar inte antal kolumner i den refererade tabellen.
2075	Okänd kolumn "%s" i foreign-nyckeldefinition.
2076	Tabellen "%s" får inte indexeras.
2077	Vyer får inte indexeras.
2080	De ON CONFLICT-satser som angetts står i konflikt med varandra.
2081	Inget sådant index: "%s"
2082	Det index som är associerat med UNIQUE- eller PRIMARY KEY-begränsningen kan inte tas bort.
2083	BEGIN tillåts inte i SQL.
2084	COMMIT tillåts inte i SQL.
2085	ROLLBACK tillåts inte i SQL.
2086	Det gick inte att öppna en temporär databasfil för att lagra temporära tabeller.
2087	Det gick inte att identifiera objektet som skulle omindexeras.
2088	Tabellen "%s" får inte ändras.
2089	Det gick inte att ändra "%s" eftersom det är en vy.
2090	Variabelvärdet måste ligga mellan ?0 och ?%d.
2092	Felanvändning av sammanställningsalias "%s".
2093	Tvetydigt kolumnnamn: "[%s.[%s.]]%s"
2094	Ingen sådan funktion: "%s"

2095	Felaktigt antal argument till funktionen "%s".
2096	Underfrågor tillåts inte i CHECK-begränsningar.
2097	Parametrar tillåts inte i CHECK-begränsningar.
2098	Uttrycksträdet är för stort (max djup %d).
2099	RAISE() får bara användas i ett utlösar-program.
2100	Tabellen "%s" har "%d" kolumner men %d-värden angavs.
2101	Databasschemat är låst: "'%s"
2102	Programsatsen är för lång.
2103	Det gick inte att ta bort/ändra kollationssekvensen på grund av aktiva programsatser.
2104	För många kopplade databaser - max %d.
2105	Det går inte att göra ATTACH på databas inom transaktionen.
2106	Databasen "%s" används redan.
2108	Kopplade databaser måste använda samma textkodning som huvuddatabasen.
2200	Slut på minne.
2201	Det gick inte att öppna databasen.
2202	Det går inte att utföra DETACH på en databas under en transaktion.
2203	Det går inte att koppla från databasen: "%s"
2204	Databasen "%s" är låst.
2205	Det gick inte att låsa databasen för läsning.
2206	[kolumnen|kolumnerna] "%s[,%s]" är inte [unik|unika].
2207	Felformat databasschema.
2208	Filformatet stöds inte.
2209	Okänd token: "%s"
2300	Det gick inte att konvertera textvärdet till ett numeriskt värde.
2301	Det gick inte att konvertera strängvärdet till ett datum.
2302	Det gick inte att konvertera flyttalsvärdet till ett heltal utan dataförluster.
2303	Det gick inte att återställa transaktionen - SQL-satser bearbetas.
2304	Det gick inte att genomföra transaktionen - SQL-satser bearbetas.
2305	Databastabellen är låst: "'%s"
2306	Skrivskyddad tabell.
2307	Strängen eller BLOB-objektet är för stor/stort.
2309	Det gick inte att öppna indexerad kolumn för skrivning.
2400	Det gick inte att öppna värdet av typen %s.
2401	Ingen sådan radidentifierare: %s
2402	Objektnamnet är reserverat för interna syften: "%s"
2403	Vyn "%s" får inte ändras.
2404	Standardvärdet för kolumnen "%s" är inte en konstant.
2405	Saknar behörighet att använda funktionen "%s".
2406	Fel användning av sammanställningsfunktionen "%s".
2407	Fel användning av sammanställningen: "%s".
2408	Ingen sådan databas: "%s"
2409	Tabellen "%s" har ingen kolumn med namnet "%s".
2501	Ingen sådan modul: "%s"
2508	Ingen sådan savepoint: "%s"
2510	Det gick inte att återställa – ingen transaktion är aktiv.
2511	Det gick inte att utföra – ingen transaktion är aktiv.

Kapitel 68: Adobe Graphics Assembly Language (AGAL)

AGAL (Adobe Graphics Assembly Language) är ett skuggningsspråk som används för att definiera vertex- och fragment-återgivningsprogram. AGAL-programmen måste överföras till återgivningssammanhanget i det binära bytekodsformatet som beskrivs i detta dokument.

Bytekodsformatet AGAL

För AGAL-bytekod måste formatet Endian.LITTLE_ENDIAN användas.

Bytekodsrubrik

AGAL-bytekoden måste börja med en sjubytesrubrik:

```
A0 01000000 A1 00 -- for a vertex program
A0 01000000 A1 01 -- for a fragment program
```

Förskjutning (byte)	Storlek (byte)	Namn	Beskrivning
0	1	magic	måste vara 0xa0
1	4	version	måste vara 1
5	1	ID för skuggningstyp	måste vara 0xa1
6	1	skuggningstyp	0 för ett vertex-program; 1 för ett fragment-program

Token

Rubriken följs av ett godtyckligt antal token. Varje token är 192 bitar (24 byte) och har alltid formatet:

```
[opkod] [mål] [källa1] [källa2 eller sampler]
```

Alla opkoder använder inte alla dessa fält. Oanvända fält måste vara 0.

Åtgärdskoder

Fältet [opkod] är 32 bitar och kan anta ett av dessa värden:

Namn	Opkod	Åtgärd	Beskrivning
mov	0x00	move	flytta från källa1 till mål, komponentvis
add	0x01	lägg till (add)	mål = källa1 + källa2, komponentvis
sub	0x02	minus (subtract)	mål = källa1 - källa2, komponentvis
mul	0x03	multiplicera	mål = källa1 * källa2, komponentvis
div	0x04	dividera	mål = källa1 / källa2, komponentvis

Namn	Opkod	Åtgärd	Beskrivning
rcp	0x05	reciprokt (reciprocal)	mål = 1/källa1, komponentvis
min	0x06	minimum	mål = minimum(källa1, källa2), komponentvis
max	0x07	maximum	mål = maximum(källa1, källa2), komponentvis
frc	0x08	bråk (fractional)	mål = källa1 - (float)floor(källa1), komponentvis
sqt	0x09	kvadratrot (square root)	mål = sqrt(källa1), komponentvis
rsq	0x0a	reciprokt rot (reciprocal root)	mål = 1/sqrt(källa 1), komponentvis
pow	0x0b	potens (power)	mål = pow(källa1, källa2), komponentvis
log	0x0c	logaritm	mål = log_2(källa1), komponentvis
exp	0x0d	exponent	mål = 2^källa1, komponentvis
nrm	0x0e	normalize	mål = normalize(källa1), komponentvis (resulterar endast i ett 3-komponentsresultat, mål måste maskeras till .xyz eller mindre)
sin	0x0f	sinus	mål = sin(källa1), komponentvis
cos	0x10	cosinus	mål = cos(källa1), komponentvis
crs	0x11	summaprodukt (cross product)	mål.x = källa1.y * källa2.z - källa1.z * källa2.y mål.y = källa1.z * källa2.x - källa1.x * källa2.z mål.z = källa1.x * källa2.y - källa1.y * källa2.x (Resulterar endast i ett 3-komponentsresultat, mål måste maskeras till .xyz eller mindre)
dp3	0x12	punktprodukt (dot product)	mål = källa1.x*källa2.x + källa1.y*källa2.y + källa1.z*källa2.z
dp4	0x13	punktprodukt (dot product)	mål = källa1.x*källa2.x + källa1.y*källa2.y + källa1.z*källa2.z + källa1.w*källa2.w
abs	0x14	absolut	mål = abs(källa1), komponentvis
neg	0x15	negera	mål = källa1, komponentvis
sat	0x16	saturnera	mål = maximum(minimum(källa1,1),0), komponentvis
m33	0x17	multipliceringsmatris 3x3	mål.x = (källa1.x * källa2[0].x) + (källa1.y * källa2[0].y) + (källa1.z * källa2[0].z) mål.y = (källa1.x * källa2[1].x) + (källa1.y * källa2[1].y) + (källa1.z * källa2[1].z) mål.z = (källa1.x * källa2[2].x) + (källa1.y * källa2[2].y) + (källa1.z * källa2[2].z) (Resulterar endast i ett 3-komponentsresultat, mål måste maskeras till .xyz eller mindre)

Namn	Opkod	Åtgärd	Beskrivning
m44	0x18	multipliceringsmatris 4x4	mål.x = (källa1.x * källa2[0].x) + (källa1.y * källa2[0].y) + (källa1.z * källa2[0].z) + (källa1.w * källa2[0].w) mål.y = (källa1.x * källa2[1].x) + (källa1.y * källa2[1].y) + (källa1.z * källa2[1].z) + (källa1.w * källa2[1].w) mål.z = (källa1.x * källa2[2].x) + (källa1.y * källa2[2].y) + (källa1.z * källa2[2].z) + (källa1.w * källa2[2].w) mål.w = (källa1.x * källa2[3].x) + (källa1.y * källa2[3].y) + (källa1.z * källa2[3].z) + (källa1.w * källa2[3].w)
m34	0x19	multipliceringsmatris 3x4	mål.x = (källa1.x * källa2[0].x) + (källa1.y * källa2[0].y) + (källa1.z * källa2[0].z) + (källa1.w * källa2[0].w) mål.y = (källa1.x * källa2[1].x) + (källa1.y * källa2[1].y) + (källa1.z * källa2[1].z) + (källa1.w * källa2[1].w) mål.z = (källa1.x * källa2[2].x) + (källa1.y * källa2[2].y) + (källa1.z * källa2[2].z) + (källa1.w * källa2[2].w) (Resulterar endast i ett 3-komponentsresultat, mål måste maskeras till .xyz eller mindre)
kil	0x27	avbryt/förkasta (endast fragmentskuggning) (kill)	Om en skalärkällkomponent är mindre än noll, förkastas fragment och dras inte till bildrutebufferten. (Målregistret måste vara 0)
tex	0x28	texturexempel (endast fragmentskuggning)	destination lika med inläsning från textur källa2 vid koordinaterna källa1. I detta fall måste källa2 ha ett sampler-format
sge	0x29	ange om större än eller lika med (set-if-greater-equal)	mål = källa1 >= källa2 ? 1 : 0, komponentvis
slt	0x2a	ange om mindre än (set-if-less-than)	mål = källa1 < källa2 ? 1 : 0, komponentvis
seq	0x2c	ange om lika med (set-if-equal)	mål = källa1 == källa2 ? 1 : 0, komponentvis
sne	0x2d	ange om inte lika med (set-if-not-equal)	mål = källa1 != källa2 ? 1 : 0, komponentvis

Format för målfält

Fältet [mål] är 32 bitar stort:

```
31.............................0
----TTTT----MMMMNNNNNNNNNNNNNNNN
```

T = Registertyp (4 bitar)

M = Skrivmask (4 bitar)

N = Registernummer (16 bitar)

- = odefinierad, must be 0

Format för källfält

Fältet [källa] är 64 bitar stort:

```
63.............................................................0
D------------QQ----IIII----TTTTSSSSSSSSOOOOOOOONNNNNNNNNNNNNNNN
```

D = Direkt=0/Indirekt=1 för direkt Q och I ignoreras, 1 bit

Q = Val av indexregisterkomponent (2 bitar)

I = Indexregistertyp (4 bitar)

T = Registertyp (4 bitar)

S = Swizzle (8 bitar, 2 bitar per komponent)

O = Indirekt förskjutning (8 bitar)

N = Registernummer (16 bitar)

- = odefinierad, must be 0

Format för samplerfält

Det andra källfältet för tex-opkoden måste ha formatet [sampler], som är 64 bitar stort:

```
63.............................................................0
FFFFMMMMWWWWSSSSDDDD--------TTTT--------BBBBBBBBNNNNNNNNNNNNNNNN
```

N = Samplerregisternummer (16 bitar)

B = LOD-avvikelse (level-of-detail) för textur, heltal med tecken, skala med 8. Flyttalet som används b/8.0 (8 bitar)

T = Registertyp, måste vara 5, sampler (4 bitar)

F = Filter (0=närmast, 1=linjär) (4 bitar)

M = Mipmap (0=inaktiverad, 1=närmast, 2=linjär)

W = Brytning (0=begränsning, 1=upprepa)

S = Specialflaggsbitar (måste vara 0)

D = Dimension (0=2D, 1=Kub)

Programregister

Följande registertyper är definierade. Använd det listade värdet för att ange en registertyp i fälten för en token:

Namn	Värde	Antal per fragment-program	Antal per vertex-program	Användning
Attribut	0	ej tillämpligt	8	Vertex-skuggningsindata; läs från en vertex-buffert angiven med Context3D.setVertexBufferAt().
Konstant	1	28	128	Skuggningsindata; ställ in med funktionsfamiljen Context3D.setProgramConstants().
Tillfällig	2	8	8	Temporärt register för beräkning, inte åtkomligt utanför program.

Namn	Värde	Antal per fragment-program	Antal per vertex-program	Användning
Utdata	3	1	1	Skuggningsindata: i ett vertex-program är utdata klipputrymmespositionen; i ett fragment-program är utdata en färg.
Varierande	4	8	8	Överför interpolerade data mellan vertex- och fragment-skuggningsprogram. De varierande registren från vertex-programmet tillämpas som indata för fragment-programmet. Värden interpoleras enligt avståndet från triangelhörnpunkterna.
Sampler	5	8	ej tillämpligt	Fragment-skuggningsindata; läs från en textur angiven med Context3D.setTextureAt().